

版权注意事项：1、书籍版权归著者和出版社所有；
2、本PDF仅用于个人获取知识，进行私底下知识交流；
3、PDF获得者不得在互联网以任何目的进行传播；
如有需要，请尽量购买正版实体书！支持书籍作者！！

企业级应用性能管理最佳实践
时刻保持最佳用户体验



大型网站性能 监测、分析与优化

唐文◎著



中国工信出版集团

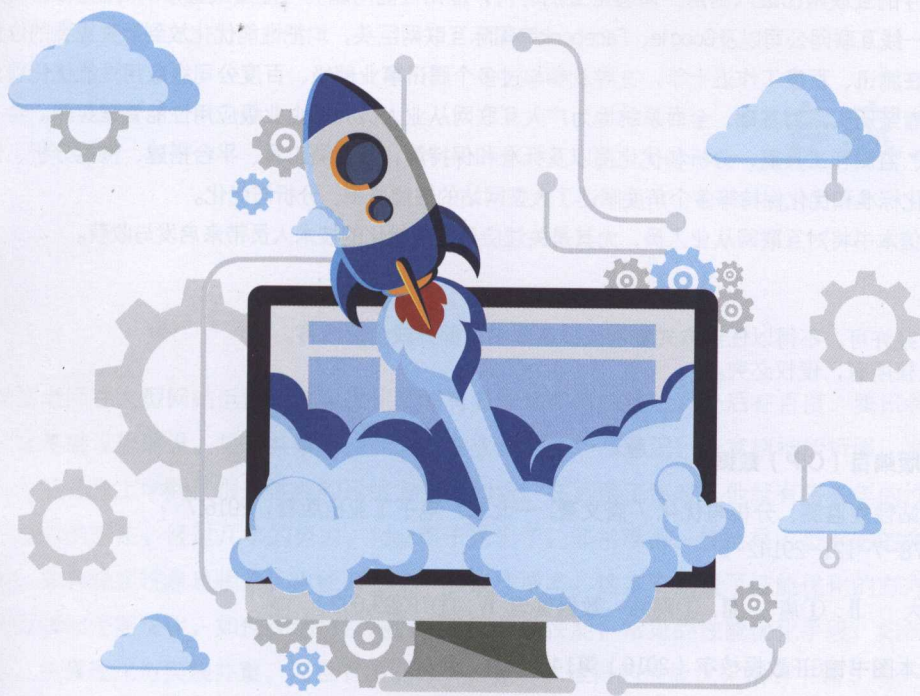


电子工业出版社
PUBLISHING HOUSE OF ELECTRONICS INDUSTRY
<http://www.phei.com.cn>

作者简介



唐文，曾负责腾讯四大平台之一网络媒体平台的整体运维、运营规划工作；曾任百度 T7 架构师和百度性能优化 TOPIC、百度 UAQ、APM 平台负责人；畅销书《海量运维、运营规划之道》作者；mmTrix 创始人，后并入上市公司高升控股（000971.SZ），出任技术 VP。



大型网站性能 监测、分析与优化

唐文◎著

电子工业出版社

Publishing House of Electronics Industry

北京·BEIJING

内容简介

当今的互联网已进入到了用户体验至上的时代,应用性能问题有可能造成直接的商业损失,以BAT为代表的一线互联网公司以及Google、Facebook等国际互联网巨头,均把性能优化放到至关重要的位置。作者唐文在腾讯、百度工作近十年,主导及参与过多个腾讯事业部级、百度公司级应用性能优化项目,本书以作者亲历实践为基础,全面系统地为广大互联网从业人员介绍企业级应用性能管理实践。全书分为基础篇、监测和工具篇、分析和优化篇以及标准和保持篇,从监测工具、平台搭建、性能分析、性能优化、优化标准和优化保持等多个角度阐述了大型网站的性能监测、分析与优化。

相信本书将对互联网从业人员,尤其是关注应用性能优化的技术人员带来启发与收获。

未经许可,不得以任何方式复制或抄袭本书之部分或全部内容。
版权所有,侵权必究。

图书在版编目(CIP)数据

大型网站性能监测、分析与优化 / 唐文著. —北京: 电子工业出版社, 2016.7
ISBN 978-7-121-29142-5

I. ①大… II. ①唐… III. ①网站—数据管理 IV. ①TP393.092

中国版本图书馆CIP数据核字(2016)第140328号

策划编辑: 张月萍

责任编辑: 徐津平

印刷: 中国电影出版社印刷厂

装订: 三河市良远印务有限公司

出版发行: 电子工业出版社

北京市海淀区万寿路173信箱

邮编: 100036

开本: 787×980 1/16

印张: 21 字数: 477千字

版次: 2016年7月第1版

印次: 2016年7月第1次印刷

印数: 4000册 定价: 88.00元

凡所购买电子工业出版社图书有缺损问题, 请向购买书店调换。若书店售缺, 请与本社发行部联系, 联系及邮购电话: (010) 88254888, 88258888

质量投诉请发邮件至zlts@phei.com.cn, 盗版侵权举报请发邮件至dbqq@phei.com.cn。

本书咨询联系方式: 010-51260888-819 faq@phei.com.cn。

专家力荐

唐文老师在大型网站运维和性能优化领域有着非常丰富的经验，先后在百度、腾讯参与运维及性能优化基础设施建设。与他共事时，我为其专业素养、产品意识、分享精神所折服，在交流合作的过程中从他身上学到了很多性能和运维方面的知识。在百度工作时，他就有将多年的性能优化经验汇总成书的想法，经过几年的努力，idea终于兑现了，非常难得。性能是大型网站的命脉，与用户体验、服务稳定性息息相关，也能为企业节省运作成本。这本书涵盖了性能优化的方方面面，包括为什么要做性能优化，如何评估与监测应用及服务的性能，常见的性能优化手段，如何防止性能退化等。书籍理论与实践并重，而且有案例分析，是一本难得的性能优化指导书籍。

——百度前端性能优化负责人，牛亮

在互联网环境日益复杂的今天，性能优化的价值在不断提升。从用户体验到成本，各个方面都可产生收益，尤其在大型项目上这种收益更是呈现指数级的增加；另一方面，找到与竞争对手的性能差距，也是帮助产品超越对手的重要手段，性能的本质提升将大大助力项目突破难关。

性能优化是一个大型的系统性工程，云、管、端各个层面技术栈全面，从检测分析到优化落地，本书体现了作者在这个领域的丰富经验。唐文在B公司时我与他一起共事，同在上海张江的那几年时光，见证了他通过对不同场景下的性能优化给公司产品带来的收益。他的第一本原创书籍《海量运维、运营规划之道》在领域内创下了记录，而本次又是原创，且在性能优化这个领域，非常的难得和期待。

——百度运维部高级架构师，王炜煜

应用性能管理当前已成为互联网非常火爆的话题，其内容涵盖了与用户体验息息相关的系统监测、网络服务、应用代码、性能优化等环节。唐文先生在互联网公司的长期性能管理从业经验为本

书注入了科学化、系统化、专业化的应用性能管理实践视角。随着移动互联网的不断发展壮大，国内的应用研发体系也在不断地健全并趋于成熟化，应用性能管理将逐渐成为应用研发体系中不可或缺的一环，并为终极的用户体验保驾护航。衷心希望本书能够引导并帮助国内的开发者养成更系统化的应用、网站研发意识与习惯。

——阿里巴巴集团高级技术专家，杨镭，花名冷茗

在互联网行业中，速度是决定能否成功，甚至能否生存下去的关键因素。一方面需要将一个好的idea快速产品化，抢占市场。另一方面则是要保证服务的持续可用、高效运行。有时候我们会有这样的疑问，同样的IDC、网络设备、服务器配置，甚至别人的系统架构比自己的还要复杂，为什么自己网站的访问速度不如别人？其实这也是很多大型互联网公司要解决的问题。他们除了对比竞品的功能差异，也会对比效率上的差异。且运行效率一直是大厂追求的重要指标，并不断进行突破。

网站性能的提升是一个复杂且系统化的工程，包括用户信息收集、系统信息收集、衡量指标设定、综合分析、优化方法、效果评估等很多环节和技术细节。特别是随着移动互联网技术和移动设备的快速发展，终端的多样性、系统环境的复杂性、地理位置的不确定性，等等，都使得此项工作更具挑战。

本书作者是我多年的同事。在百度期间，专注于用户访问质量与效率提升方面的工作，并取得了斐然的成绩。作者希望将多年积累的工作经验进行系统化整理，抽象出一定的方法，让更多工程师系统地了解整个调优的过程及如何保持高效，最终让更多互联网企业受益。

——小米资深架构师，伏晖

《大型网站性能监测、分析与优化》是一本极具学习与参考价值的书。这本书详细阐述了性能优化领域的全链条环节，也通过作者在多家一线互联网公司的真实项目实践与思考一一展示。

有幸与本书作者唐文先生在腾讯期间有过几个大型项目的合作，一直保持着技术维度的长期学习交流。同时对于他在海量运营、业务优化、工具规划的全局观印象深刻。此书也是继他第一本书《海量运维、运营规划之道》出版2年后的又一部运维巨著，推荐阅读！

——京东无线总监 徐奇琛

2012年认识作者，那时候他刚刚从腾讯过来百度，我们有幸一同工作过两个月，之后便见证了他全身投入到APM领域并成为领先探索者的过程。作者在腾讯将网站的访问速度提升数倍，在百

度让UAQ系统成为支撑产品性能优化的核心平台。我是在今年春天去杭州的火车上收到新书的初稿，内容之丰富，让我赞叹。整书的编排也非常合理，结构清晰翔实。既是新手入门参考，也可作为性能工程师的实用手册。在当今全面移动化的互联网技术背景下，《大型网站性能监测、分析与优化》这本书一定是了解有关网站性能的图书首选。

——美丽说运维总监，何威

互联网已经全面进入用户体验至上的时代，直接决定用户体验的应用性能优化能力，也已成为国际和国内各大互联网公司的核心竞争力，直接影响着公司效益和营收。通过优化，不但能够提升资源利用率和降低资源成本，同时还能够提升用户体验，为网站带来更多流量并增加用户的黏性，从而为公司带来更多业务收入，可谓一举多得。但是，用户体验优化是一个复杂的系统性工程，对技能深度和知识面广度的要求极高。唐文结合在腾讯、百度大规模性能优化的实战经历和经验，全面、系统、细致地总结了这样一套互联网企业级性能监测、分析和优化方法论，浓缩成书。我想这本书一定可以为奋战在运维一线的小伙伴们厘清优化思路、全面提升自身技能并找到新的发展方向起到极大的指引作用，值得一读。

——蘑菇街运维负责人，赵成

总体来说，稳定性和速度是最基本的体验保障。对于技术人员来说，系统性能的监控和优化是必备技能之一。本书作者唐文在这方面有着非常丰富的实践经验，感谢他能够把这些经验总结出来并分享给大家，让我们一起来品尝这顿技术大餐吧！

——车易拍运维总监，赵旭

当我看到本书的初稿时，顿觉眼前一亮，里面的内容非常翔实，组织得非常成体系，实战性强。本书的作者唐文在访问速度监控和优化上有很多成功的经验和成体系的方法，在本书中更是进行了完整的梳理和总结。相信通过此书能够给读者带来更多的新知识、新方法、新视野，并能快速应用到日常工作中。相信读了此书的朋友一定会和我有同样的感受。

——58赶集集团高级技术经理，龚诚

初识唐文，是他给我们做互联网运维专题的专家讲座，说是专家，却看到的是一个年轻文静的小伙，十多年的互联网工作经验，做过好几个大型项目并取得不错的成果，开放、坦诚，不但精于技术，而且对技术支撑的商业价值很敏锐，文文静静的还有点小幽默，那个猫捉老鼠的故事让我们

哈哈大笑，当时就留下了很深的印象。相通的是对技术的痴迷，对做什么事就要坚持做到最好的工匠精神的执着。

性能影响用户体验，这个毋庸置疑，搜索速度每延迟100ms，无点击比例就会增大1%，网页切换速度、操作响应时间等道理相同，互联网应用的性能影响用户体验并最终直接影响收益。对于大型网站，性能指标有很多，涉及移动、前端、后端、网络、系统、应用、软件、硬件等方方面面，如何站在用户角度，借助系统的工程方法，抓住影响用户体验的核心指标、关键环节，辅以对应的工具对相关环节性能进行监测、分析和优化，通过点滴改进逐步提升用户体验支撑产品商业成功？想想，挺不容易。

埋头干活的人不少，能在干好活之后，抬起头来，总结整理经验并上升到方法论，帮助别人，促进行业共同进步的同时也是对自己知识点的打结，结多了，自然形成了网。我默默地敬佩这样的技术专家。

章选

前言

1. 写在最前面

为什么要写这本书

在人际关系中，良好的第一印象至关重要。人们愿意在彼此身上寻求信任与安全感，并期望在接下来的实践经历中重现及增强这些好感。同样的道理也体现在互联网产品中，用户体验扮演着极其重要的角色。如今当我们访问的移动应用出现速度慢、图片页面无法打开、视频无法下载、交易拥塞等症状时，大多数人会选择毫不犹豫地离开。正是由于越来越多的互联网应用承载着企业的商业价值，而每当出现应用性能问题时便直接转化为实际收益损失，所以解决性能问题之关键，关乎用户体验，也直接影响效益营收及企业核心竞争力。

正因如此，Google、Yahoo!、Facebook等全球最优秀的互联网公司投入巨大的人力、物力，长期对性能进行优化和保持，同时也印证了高性能网站能够增加流量、提高用户体验，最终增加业务收入、降低运营成本，并沉淀下来大量行之有效的经验和工具。笔者在腾讯、百度工作近十年，不断向国际优秀前辈学习和借鉴，主导及参与多个腾讯事业部级、百度公司级应用性能优化项目，并取得巨大收益。此书以笔者亲历实践为基础，全面系统地为广大互联网从业人员介绍企业级应用性能管理实践。

这本书的不同之处

从Web 1.0时代至今，工程师们一直在尝试各种方法，提高用户对互联网产品的浏览体验。在

这个过程中，对用户体验改善有许多的理解和称呼，例如Web前端性能优化、Web性能优化、速度优化、系统性能优化、访问质量优化、可用性优化等，而无论是在哪一个时代、或是哪一种理解，都有一个共同目的——就是让用户体验更好。性能优化界的泰山北斗级人物Steve Souders曾说过一个理论：“80%~90% of the end-user response time is spent on the frontend”最为业界所认可。他是Google Web性能布道者和前Yahoo!首席性能工程师，也引领这一领域多年。他把用户体验优化基本定义为“前端+Web性能优化”。随着移动互联网的发展，这一领域逐渐延伸到移动Web性能优化领域。如今大多数工程师们基本都思考过或曾经接触过性能优化方面的知识，如《雅虎34条黄金守则》，抑或是从众所周知的那几本很经典的国外性能优化指导书中——《高性能网站建设指南》《高性能网站建设进阶指南》等。工作经验丰富的工程师们对于前端性能优化方式耳濡目染，基本都能一一列举出来。虽然这些性能优化原则大多是早些年所提出，主要偏前端范畴，但却对Web性能优化至今都有着非常重要的指导意义。

在互联网产品多样性、复杂化的今天，全球化、移动化、多终端、海量用户数据和实时性等新特点为改善用户体验带来了更大的挑战，甚至可以毫不夸张地说今天的互联网时代是比历史上任何时期都面临更加复杂的局面和更为严峻的挑战。互联网已经进入到用户体验至上的时代。企业商业价值的高度互联网化、移动化及激烈的行业竞争等，都会让企业对用户体验的追求越来越高。与此同时，用户选择门槛也会越来越低而使其更为挑剔，加上影响用户体验的因素越来越多元化及互联网技术、网络、硬件的高速迭代，这种种原因使得前端Web性能已经不再像以前那样对用户体验起到决定性作用。尤其是在中国存在其特有的差异性，从移动、前端、后端、网络、系统、应用、硬件、产品逻辑等都决定了用户体验。本书作者结合在腾讯、百度大规模性能优化的工作经历从一个全新的企业级视角来考量用户体验与工程师之间的关系，试图诠释如何建立完整企业级性能监测、分析与优化体系。

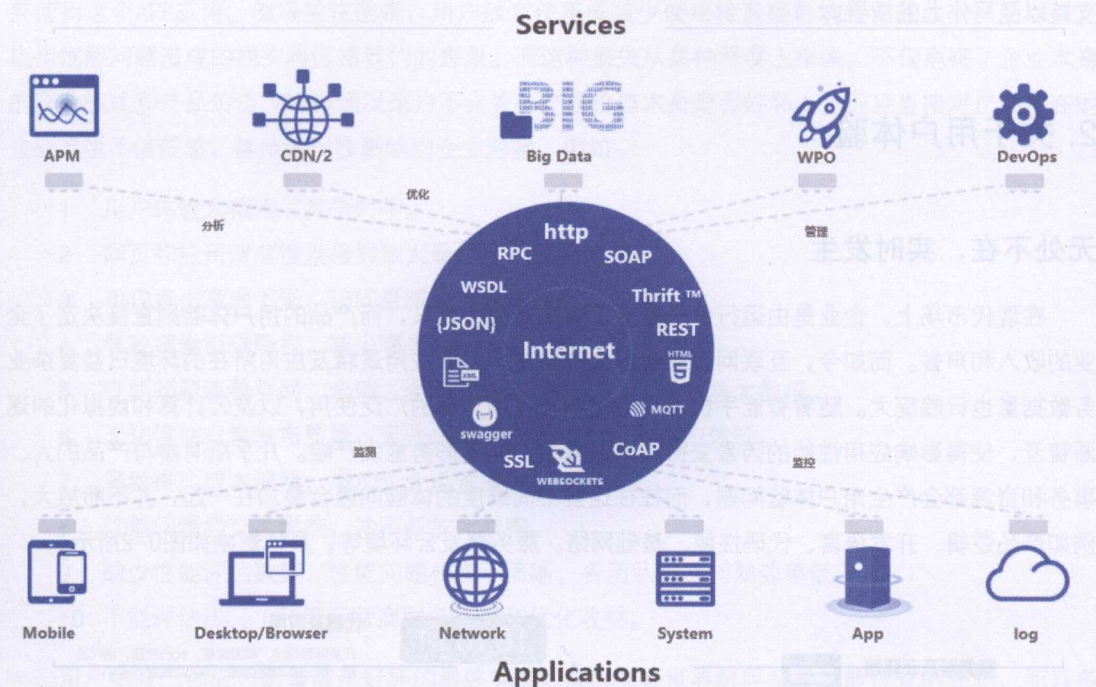


图0-1 企业级性能监测、分析、优化体系

读者对象

本书适合以下读者阅读：

互联网行业技术经理、项目经理、架构师、技术总监、CTO

互联网行业运维工程师、测试工程师、前端研发工程师、后端研发工程师、移动研发工程师

致力于从全局把握应用性能监测、分析与优化和互联网产品的所有互联网从业人员

勘误和支持

由于编写的时间仓促，书中难免会出现一些错误或者不准确的地方。恳请广大读者朋友批评指正，也欢迎您将错误和建议发送邮件至我的邮箱11599096@qq.com，期待能够听到您的真挚反馈。

致谢

感谢原百度同事朱建锋、徐晔等提供素材。最后感谢我的太太在本书写作期间给予我事业上的

支持以及写作上的指导。

2. 关于用户体验

无处不在，实时发生

在现代市场上，企业是由运行他们商业价值的产品所定义，而产品的用户体验则直接决定了企业的收入和声誉。而如今，互联网蕴藏着巨大的财富，企业应用逻辑及应用所在的环境日益复杂业务数据量也日趋庞大。随着智能手机、平板电脑与多浏览器的广泛使用，以及云计算和虚拟化的逐渐普及，使得影响应用性能的因素变得越来越复杂，并且形势愈发严峻。几乎所有参与产品的人、事务和资源都会产生用户体验问题，而往往这些不同维度的体验问题会叠加在一起，并不断放大，例如产品逻辑、开发语言、代码性能、基础网络、服务器及云环境等，具体影响如图0-2所示。

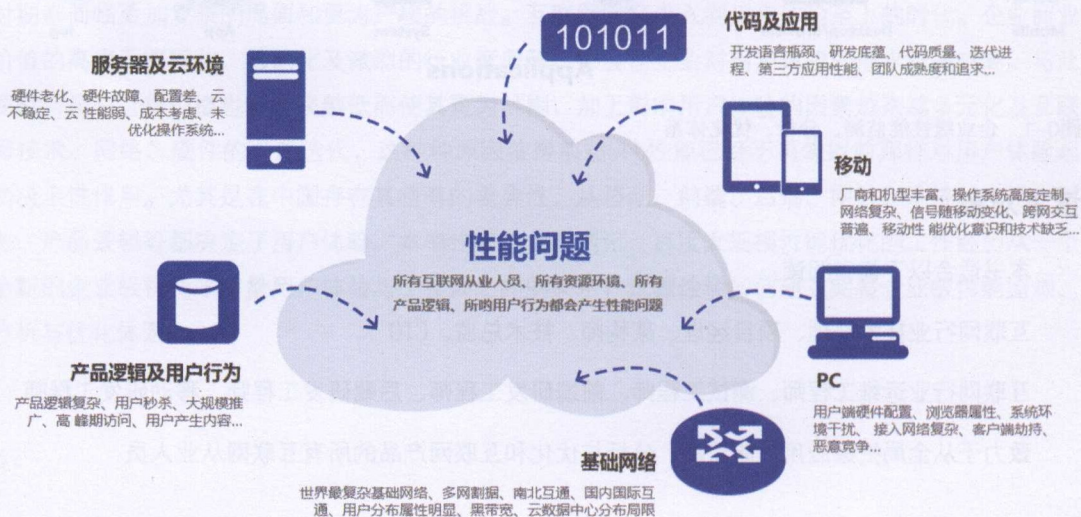


图0-2 影响应用性能的因素

直接转换为商业损失

据Google Analytics数据统计，目前移动网页平均加载时间至少需要7s；据Nielsen Norman Group的调研结果：如果移动网页加载时间超过1s，将开始影响用户的使用，导致用户产生反感。据我们自己的体验也容易理解，在手机端打开一个APP应用，如果超过3s还是白屏，我们基本会放

弃使用这个APP应用。值得关注的是，用户放弃使用或减少使用将直接影响企业的收入。所以说，应用性能问题造成的损失将远超我们的想象。而这种损失从某种程度上来说，不仅危害了企业本身的商业模式和产品价值（这种情况用户不会关心产品价值本身是否好坏），而且直接对产品所在的企业产生不信任感，甚至进一步影响到企业形象。例如：

- 1 用户体验大幅落后竞争对手。
- 2 网页和应用速度慢直接导致大量用户永久性流失。
- 3 用户点击意愿下降，访问量减少，收入锐减。
- 4 导致搜索引擎降权，减少曝光率。
- 5 性能问题随着全网、全端、全球化深入推进，损失将会放大数倍。
- 6 无法评估日常发布质量，无法保障发布是否会影响用户体验。
- 7 导致推广成本浪费，增加企业运营成本。
- 8 性能问题会交叉影响，并不断放大危害。
- 9 缺少性能评估数据、性能问题权责不清晰，各团队解决问题效率低下。
- 10 不能评估IDC、CDN等运营商服务质量和优化收益。

用户体验已经成为衡量应用好坏的最终标准。Google等世界级巨头是性能优化的先驱，而且多年将性能优化放在重要战略地位。

3. 在腾讯、百度实践的体会

天下武功，唯快不破

其实腾讯早在2006年就已经开始大规模性能优化。在当时还是门户混战的资讯时代，我有幸加入腾讯并负责腾讯网的整体运维和运营规划工作。由于历经两次大规模性能优化，当时的资源和工具都极其匮乏。于是乎，从组建全国分布式IDC、CDN、GSLB、质量监测等平台建设到联合8大部门近50人的跨部门性能优化团队，由前端、后端、系统、网络、内容等维度进行了体系全面的优化，使其最终反超传统门户网站，而这些平台后来也慢慢沉淀为腾讯的基础公共平台。

由于资讯时代的互联网新闻比竞争对手快一秒发布，都会抢占商业先机。所以当时互联网最核心的公司文化是用户体验第一——即快速发布、快速分发、快速打开。我们会根据实时统计的全网用户浏览新闻速度和用户偏好，随时调整、优化内容，让用户阅览新闻快如闪电。用户体验也已经成为工程师的核心文化。“速度影响用户体验”这是当时身边所有同事都默认的基本规约，也是工

作的基础准则。从架构师全局可用性设计，前端工程师优化代码、后端工程师优化逻辑、网络工程师优化延时、系统工程师优化内核、产品工程师优化体验等，每位同事都在为用户体验做贡献。而我有幸经历了腾讯网两次大规模性能优化，使其从原来远远落后业界水平，四大门户最差，到经过优化后全面反超，具体优化与收益如图0-3所示，由此带领团队也获得了腾讯重大架构一等奖。

经过三年两次大规模的性能优化，让自己对互联网职业有了更为深刻的理解。任何职业，无论是产品、研发还是测试、运维，越往后发展，衡量标准不仅仅是职业基本技能，而能否考量用户体验已经成为优秀与否的重要标准之一。用户体验的优化更是一个复杂的系统性工程，需要一套完整的监测、分析、优化平台和方法论，而这更迫切地需要每一个参与产品的人在意识上的高度重视，任重且道远。

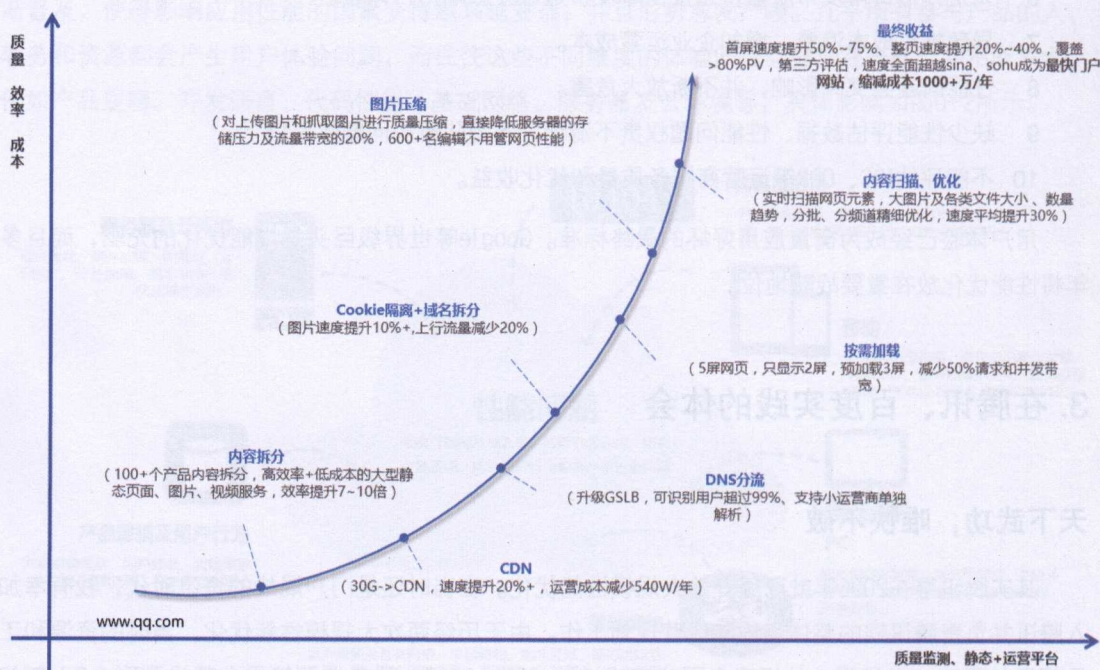


图0-3 速度优化成功案例

搜索之争，一秒判生死

自2011年加入百度，我主要负责百度产品的访问速度优化改进。当时中国互联网已经进入“搜索时代”，让人们最平等便捷地获取信息、找到所求，是百度一直孜孜不倦的追求。而搜索速度一直是搜索引擎用户体验重要的组成部分，也是技术上挑战最大的难题之一。一方面，搜索引擎趋向于索引越来越大的数据和采用越来越复杂的策略算法，这些都会增加后端检索时间，导致搜索速度

变慢；另一方面，用户电脑性能和网络环境都对搜索速度有非常大的影响，而这些因素都在搜索引擎技术控制的范围之外。

那么每提高0.5s的检索速度需要多大的投入？事实上，通过极速搜索的创新模式，在保证搜索质量的同时，百度能够将原有搜索速度提高5~10倍，最快0.04s即能返回结果。而为了实现这一功能，30多位百度核心工程师对检索系统的29种技术模块进行了升级改造，保证了极速搜索服务的可靠性和稳定性。为了解决预测所需的大量计算，百度全年在服务器等基础设施上投入超过一亿元人民币。那么每提升0.5s的速度，又能给用户和百度带来多么大的价值？百度曾做过一个有关搜索速度和用户体验关系之间的试验：在一个较小的区间内，搜索速度每延迟100ms，无点击比例就会增大1%；而随着区间的增长，这个关系会呈现出指数级的曲线。由此可见，虽然我们无法明显感知100ms的变化，但眼球已按照潜意识的指引做出选择。正因如此，可以说，速度是令搜索引擎高下立判，甚至决定其成败的要穴。有鉴于此，历经近三年的持续性能优化改进，最终将百度网页搜索、移动搜索、多个商业产品及社区产品速度优化到业界最快。

由于当时没有成体系的监测工具，第三方只能满足很小的一部分需求。而要支撑公司级性能优化，每年要支付给第三方300万元。最终我带领团队搭建了UAQ（用户访问质量）、APM（应用性能管理）平台，协助百度网页搜索、移动搜索、多个商业产品及社区产品速度优化到业界最快，这些平台通过融入百度商业平台，直接让百度的企业客户受益，如图0-4和图0-5所示。

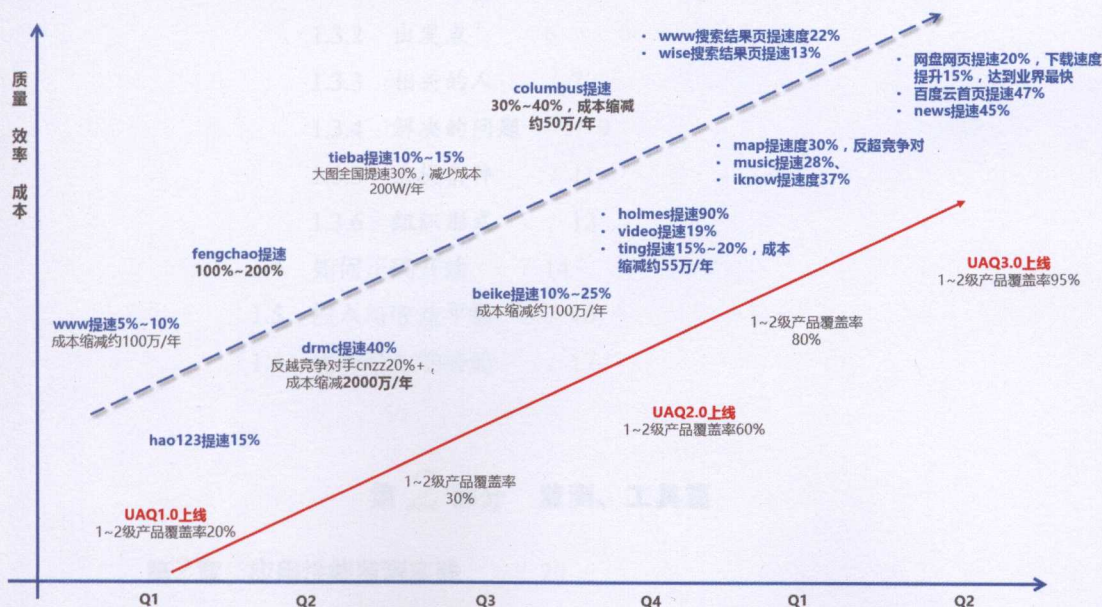


图0-4 百度性能优化及收益

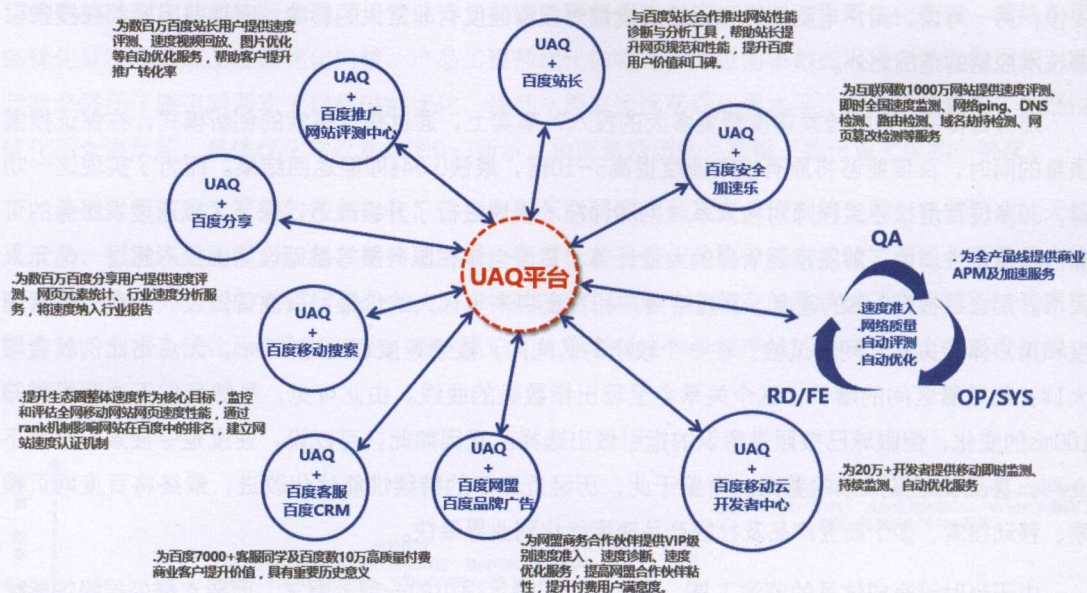


图0-5 性能优化对内、对外价值参考

目录

第1部分 基础篇

第1章 应用性能管理概述 / 2

- 1.1 关于应用性能 / 2
- 1.2 关于应用性能管理 / 2
- 1.3 基本意识 / 3
 - 1.3.1 价值与意义 / 4
 - 1.3.2 出发点 / 6
 - 1.3.3 相关的人 / 7
 - 1.3.4 解决的问题 / 9
 - 1.3.5 前提条件 / 11
 - 1.3.6 组织形式 / 12
- 1.4 如何正确开始 / 14
- 1.5 投入与收益平衡 / 16
- 1.6 优秀企业的经验 / 17

第2部分 监测、工具篇

第2章 应用性能监测实践 / 20

- 2.1 应用性能监测概述 / 20

2.2	应用性能持续监测	/ 23
2.2.1	移动监测	/ 24
2.2.2	Web监测	/ 37
2.2.3	系统监测	/ 56
2.2.4	应用监测	/ 62
2.2.5	日志监测	/ 89
2.3	应用性能即时监测	/ 94
2.3.1	PC即时监测	/ 101
2.3.2	移动Web App即时评测	/ 104
2.3.3	移动Native App即时评测	/ 106
2.3.4	网络即时监测	/ 109

第3章 性能监测工具介绍 / 113

3.1	监测工具概述	/ 113
3.2	持续监测工具	/ 115
3.2.1	Keynote	/ 116
3.2.2	Dynatrace	/ 117
3.2.3	App dynamics	/ 119
3.2.4	Newrelic	/ 120
3.2.5	基调	/ 122
3.2.6	博睿	/ 124
3.2.7	OneAPM	/ 125
3.2.8	云智慧	/ 128
3.3	即时监测工具	/ 130
3.3.1	YSlow	/ 130
3.3.2	Pagespeed Insights	/ 131
3.3.3	WebPageTest	/ 132
3.3.4	ChromeDevTools	/ 133
3.3.5	PhantomJS	/ 135
3.3.6	Jspdf	/ 136
3.4	其他工具	/ 136

- 3.5 应用性能指标 / 140
 - 3.5.1 用户指标 / 140
 - 3.5.2 服务器指标 / 147
 - 3.5.3 移动指标 / 149
 - 3.5.4 其他指标 / 150

第4章 性能监测平台搭建实践 / 152

- 4.1 为什么要搭建监测平台 / 152
- 4.2 如何搭建性能监测平台 / 154

第3部分 分析、优化篇

第5章 应用性能分析实践 / 160

- 5.1 产生性能问题的因素 / 160
 - 5.1.1 产品逻辑及用户行为 / 161
 - 5.1.2 中国基础网络 / 161
 - 5.1.3 PC端环境 / 163
 - 5.1.4 移动端环境 / 164
 - 5.1.5 代码及应用 / 165
 - 5.1.6 服务器及云环境 / 166
- 5.2 应用性能分析概述 / 166
 - 5.2.1 从用户及生产环境着手 / 167
 - 5.2.2 常见的分析方法 / 176
 - 5.2.3 主要分析视图 / 182
 - 5.2.4 横向对比的意义 / 202

第6章 应用性能优化实践 / 205

- 6.1 应用性能优化概述 / 205
 - 6.1.1 确保优化方向正确 / 206
 - 6.1.2 确定优化带来的收益 / 207

6.1.3	功能与性能的平衡	/ 209
6.1.4	防止过早和过度优化	/ 209
6.2	网络优化	/ 210
6.2.1	IDC优化	/ 211
6.2.2	ISP优化	/ 217
6.2.3	CDN优化	/ 221
6.2.4	BGP优化	/ 229
6.2.5	DNS优化	/ 231
6.3	系统优化	/ 234
6.3.1	压缩优化	/ 236
6.3.2	缓存优化	/ 237
6.3.3	分离优化	/ 240
6.3.4	内核优化	/ 242
6.3.5	传输优化	/ 245
6.3.6	并发优化	/ 248
6.3.7	隔离优化	/ 250
6.3.8	网卡优化	/ 251
6.3.9	硬件优化	/ 254
6.4	前端优化	/ 257
6.4.1	首屏优化	/ 259
6.4.2	内容优化	/ 261
6.4.3	请求优化	/ 263
6.4.4	CSS优化	/ 266
6.4.5	JavaScript优化	/ 269
6.4.6	图片优化	/ 271
6.5	后端优化	/ 275
6.5.1	架构优化	/ 275
6.5.2	并行优化	/ 276
6.5.3	异步优化	/ 276
6.5.4	基础优化	/ 276
6.5.5	算法优化	/ 277

6.5.6	程序优化	/ 277
6.5.7	缓存优化	/ 278
6.6	移动优化	/ 278
6.6.1	网络优化	/ 279
6.6.2	请求优化	/ 282
6.6.3	缓存优化	/ 283
6.6.4	策略优化	/ 284
6.6.5	启动优化	/ 286
6.6.6	交互优化	/ 286
6.6.7	内存优化	/ 287
6.7	其他优化	/ 290
6.7.1	SPDY	/ 290
6.7.2	HTTP/2	/ 291
6.7.3	ESI	/ 293
6.7.4	SDCH	/ 294
6.7.5	BigPipe	/ 294
6.7.6	DNS Prefetch	/ 295
6.7.7	HHVM	/ 295

第7章 性能优化平台搭建实践 / 296

7.1	为什么要搭建优化平台	/ 296
7.2	如何搭建性能优化平台	/ 297

第4部分 标准、保持篇

第8章 应用性能优化标准 / 304

8.1	防止应用性能退化概述	/ 304
8.2	通过规范防止性能退化	/ 304
8.3	通过流程防止性能退化	/ 307

- 8.3.1 应用性能准入 / 307
- 8.3.2 应用性能认证 / 308
- 8.3.3 应用性能巡检 / 309
- 8.4 业界优秀企业的经验 / 310
 - 8.4.1 雅虎Web优化最佳实践 / 310
 - 8.4.2 谷歌Web优化最佳实践 / 310

第9章 应用性能优化保持 / 313

- 9.1 性能优化保持概述 / 313
- 9.2 通过平台防止性能退化 / 313
 - 9.2.1 自动优化开发框架 / 313
 - 9.2.2 自动优化基础平台 / 314
- 9.3 通过告警防止性能退化 / 314

第1部分

基础篇

第1章 应用性能管理概述

第1章 应用性能管理概述

1.1 关于应用性能

应用性能是互联网产品的一种非功能特性，它关注的不是产品能否完成特定的功能，而是在完成该功能时展示出来的及时性。由于感受应用性能的主体是人，所以应用性能是一个可感知的综合用户体验值。这个体验值深受用户使用产品的所有因素影响，这些内在和外在的很多因素都对应用性能和可用性造成干扰，如从用户发起对应用的访问，到收到该访问反馈的内容，通常会经过DNS查询、网络传输和接入转发，以及Web服务、应用服务、中间件、数据库等应用组件的信息处理，这些组件的性能优劣，都会直接影响业务交互的实时性、准确性和稳定性，但是这些影响基本上都无法完全消除或解决，且主要发生在生产环境，难以在开发、测试环境中发现。

1.2 关于应用性能管理

其实应用性能管理由来已久，IT从业者们对它的理解与实践也几乎是从IT诞生就已开始。而最终应用性能都通过产品体现给最终用户。诚然，应用性能管理的本质是通过掌控应用性能状况，从而改善应用性能，为用户提供最好的用户体验。当下的应用性能管理已经不是传统的监控系统，也不是单纯地使用“Web最佳实践”去优化网站，而是为针对企业应用特性建立一套科学完整的应用性能监测、分析、优化体系。由于监测和优化势必将影响应用性能的所有维度，所以在生产运维环境中，一旦出现应用问题征兆，应在影响扩大之前，发现问题、判断原因、隔离故障，从而达到高效解决问题之目的。

应用性能管理是系统工程

Strangeloop的研究数据表明,网站出现1s的延迟后会导致页面转换率降低7%,流量下降11%,用户满意度降低16%。那么,在100%的网站访客中,就会有57%的访客在等待3s后放弃,其中80%访客不会再回来,50%访客转向其竞争对手网站。性能问题无时无刻不发生在我们所使用产品的所有过程和所有环境,例如PC、移动、网络、系统、应用、硬件、产品逻辑等。正因为应用性能如此重要且时刻存在,腾讯从2006年开始了大规模性能优化,并将此发展为腾讯工程师文化之一。百度、阿里在2010年后也陆续启动了大规模的性能优化项目。近几年来,越来越多的企业进行了系统性的性能优化,例如新浪、携程、美团、58同城等。正是因为以上企业对性能的不断追求,才奠定了中国互联网性能优化理论与实践基础。

应用性能管理具有门槛

在举国互联网+的时代,用户至上已经被多数企业接受。我相信企业级的应用性能管理与优化的时代即将来临,也是企业可持续保持好的用户体验的基本前提。其实像BAT级别的大型互联网企业都有专职的性能管理团队,多年进行性能管理和优化,并建立其完整的性能管理平台来协助。意旨让性能管理和优化变得更高效,并保证其产品能够给用户带来卓越并可持续的体验。事实上,如果大多数中小企业要进行性能管理和优化是有门槛的。除了需要人力投入以外,还需要实践积累的过程及可持续的维护能力。所以即便很多二三线互联网企业涉及了性能优化,在绝大多数情况下也只是产品级和部门级。而随着产品版本迭代和人员流失,性能管理随即出现退化和断层,这也是当前的基本现状。

第三方推动作用在增强

如同CDN行业一样,应用性能管理也在加速商业化脚步,即APM行业。越来越多的云服务商投入到APM行业,加快了产业化进程。随着美国排名靠前的APM厂商纷纷上市,国内APM厂商也日趋繁荣。一方面,厂商的投入及商业化手段有利于增加应用性能管理的技术水准和丰富方法论,直接降低企业落地应用性能门槛。另一方面,APM的SaaS化,使得每家企业、甚至每一个开发者都可以直接使用最优秀的APM云服务帮助改善产品性能,而无须投入额外的人力和资源来搭建应用性能管理平台。换句话说,第三方厂商的服务能力会从根本上帮助企业减少应用性能的学习成本。

1.3 基本意识

应用性能木桶理论

互联网产品是创意、研发、系统、网络、硬件、维护等所有资源相互交织的集合体,这些资源

彼此之间有着千丝万缕的联系。它们必须通过共同协作以期达到稳定产品运行及良好用户体验的最终目标。如用木桶理论帮助理解的话，也就是说一只木桶能盛多少水，并不取决于最长的那块木板，而是取决于最短的那块木板，所以也可称之为短板效应。而将木桶理论进一步延伸来看，新木桶理论认为一只木桶能装多少水，不光取决于最短的木板，更应该取决于木桶是否存有缝隙，若木桶存有缝隙，则水将不断流失。应用性能木桶理论如图1-1所示。

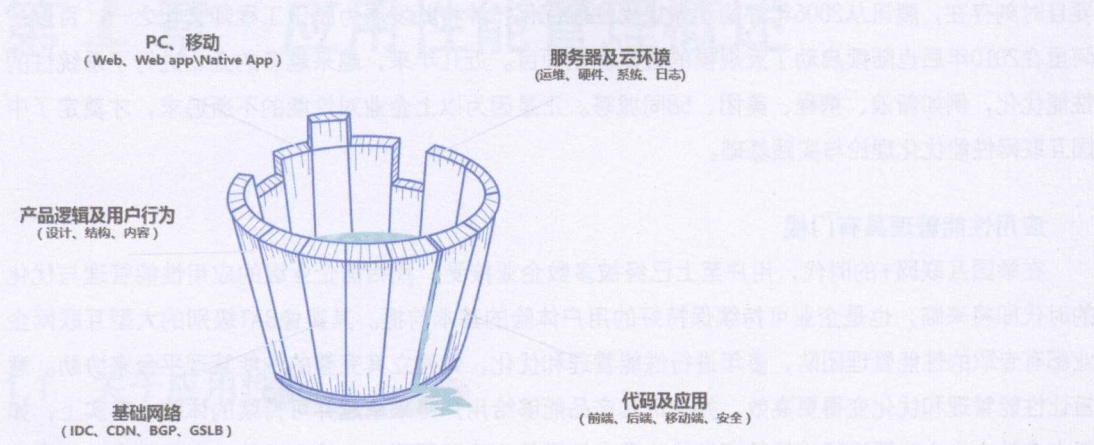


图1-1 应用性能木桶理论

基本意识和思路

应用性能监测、分析、优化的过程就是找出“木桶”的短板、缝隙并进行修复的过程。而“木桶”中的水，就是产品价值和用户体验。本人的一些相关思路，会在随后做详细阐述，以便帮助大家参考理解。

1.3.1 价值与意义

在互联网行业，要成为受人尊敬的互联网企业，产品的用户体验是一块基石——即应用性能。在腾讯工作时，常听到CEO马化腾要求所有核心产品要做到业界速度最快，并多次强调：用户放弃一个产品只需要2s，1s关掉网页，1s打开竞争对手的网页；在百度工作时，CEO李彦宏也多次强调，百度搜索80%的用户要在1s打开。事实也证明减少检索加载时间等于增加用户检索次数和广告收入。从两个中国互联网航母级的企业如此重视应用性能可以看出，应用性能对互联网产品有着至关重要的价值，而更多权威机构统计数据也从侧面说明了这一点。

速度影响用户满意度和访问量

- 1 Google的一项试验显示随着页面加载时间从400ms增至900ms，每页面搜索结果数从10增至30，将导致25%搜索者在第一个结果页放弃。

- 2 有研究显示,如果3s后,网页还未加载完毕,57%的用户会放弃。74%的用户登录某网站等待时间超过5s后就不会再登录这个网站。
- 3 有研究显示,宽带用户比窄带用户更没有耐心。宽带用户愿意忍受的最长等待时间,往往只有4~6s。60%的用户希望手机上的页面加载时间不要超过3s。

速度影响网站收益和SEO排名

- 1 Amazon的统计显示每延长1s, Amazon一年就会减少16亿美元销售额,首页打开时间每增加100ms,网站销售量会减少1%。
- 2 据估计每年电子商务网站都会因载入速度过慢,而损失11亿到13亿美元的收入。
- 3 速度在Google的PR评分中占有一定的比例,Google的PR算法中加入了速度评分一项。也就是说,一个好的网站,是不可能让它的页面加载速度很慢的。

速度是应用性能最直接体现

速度表示物体运动的快慢程度,应用或网页速度简单理解就是用户打开一个应用或网页的快慢程度。有些书也总结为浏览器向服务器发送第一个请求到最后一个网页元素加载完的消耗时间。总而言之,应用或网页加载消耗的时间越少,代表网页速度越快。根据参与腾讯、百度多个产品线优化实践的体会,通常一个网页的总加载时间 $\leq 5s$ 基本不会让用户产生反感,用户对网页第一屏加载时间 $< 2s$ 的网站会形成良好印象。图1-2所示为结合第三方公司的一些调研结果给出的用户满意度示意图,以便大家能够更好地理解应用性能。

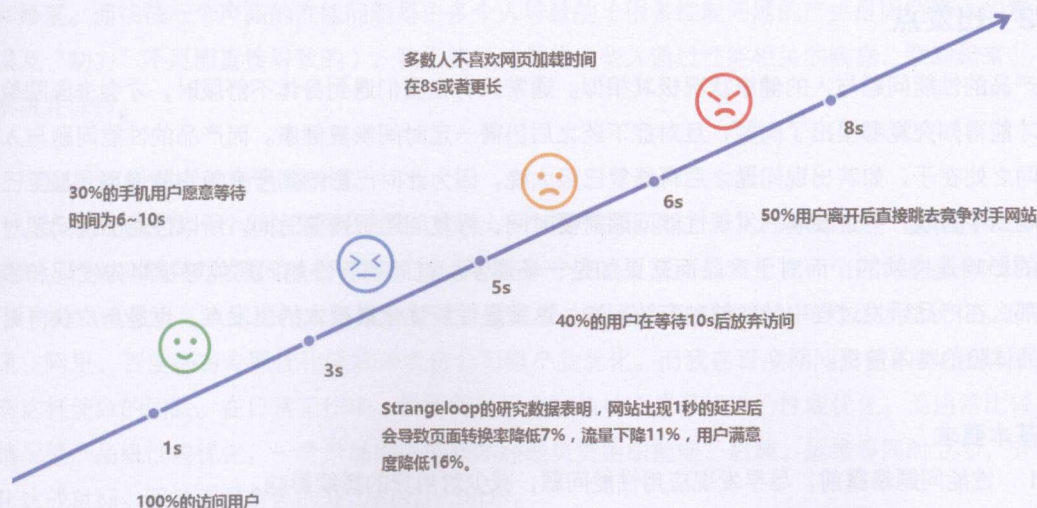


图1-2 用户满意度示意图

应用性能具有可控、可管理性

应用性能发生在产品逻辑、前端、网络、后端、硬件、系统、应用等所有环节，而这些环节之间既环环相扣又互相干扰、叠加，让性能问题更加严重。所以应用性能管理的最佳状态是让各环节平衡，并且让各环境发挥最理想的性能状态。其实就像人的健康管理一样，人在有疾病的情况下只有两个选择：要么逐渐严重并产生多种并发症让影响更严重，要么尽快就医尽快恢复减少影响。实际上，无论疾病的大小都会影响日常生活，应用性能也如此。所以应用性能管理的意义，如同健康管理的意义一样，在尽快发现和修复性能问题，并保持甚至增加产品的体验和价值上，至关重要。以下是作者在百度参与部分性能优化项目的数据，以便帮助大家理解应用性能：

- 1 百度网页搜索优化收益，包含app query 1.83s→0.61s，首屏做到极致。
- 2 百度移动云性能优化，首页2.46s→0.9s，全面反超竞品。
- 3 百度LBS速度优化，首页2.32s→1.8s，全面反超竞品。
- 4 百度移动LBS性能优化，首页2.31s→1.08s，落地页7.35s→6.25s。
- 5 百度国际化性能优化，国际化搜索速度3~8s→1~2s，国际化网址导航3.46s→2.57s。
- 6 百度贴吧frs页优化，优化后页面基本功能可用时间提升30%。
- 7 百度贴吧wap页优化，优化后页面减少40%，直接带动session上升。
- 8 百度Image改版，通过预读取提高图片展现速度，浏览量增加60%。

1.3.2 出发点

产品的性能问题与人的健康状况极其相似。通常只有在我们遇到身体不舒服时，才会去医院检查，才能得知究竟哪里出了问题，且对症下药之后仍需一定时间恢复健康。而产品的性能问题与人的不同之处在于，如若出现问题之后再修复已经太晚，因为此时已影响到所有用户的使用，甚至已经影响到了营收，并且在事后发现性能问题需要时间，修复问题更需要时间，所以性能出现问题对用户的影响是持续的，而对于产品而言更加是一场噩梦。试问如若性能问题能够被事先发现和修复，那么在产品研发过程中就能够被有效回避。这就是性能优化最根本的出发点，也是用户获得更好产品体验的基本前提。

基本要求

- 1 性能问题暴露前，尽早发现应用性能问题，减少对用户的持续影响。
- 2 性能问题发现时，尽快修复主要性能问题，减小应用性能影响力度，最大程度改进用户体验。

- 3 性能问题优化后，杜绝再次发生，可持续保持应用性能优化成果，防止性能退化。

基本原则

- 1 在产品不同生命周期性能侧重不同，优先验证简单假设，从简单到复杂，优先选择足够简单、容易出现收益的方案。
- 2 先别急着优化，优先规避性能恶化，事实和推测分开，事实验证推测，没有论证预期收益不做优化，把有限的精力投入到关键性能问题上。
- 3 从前端到后端，从外到内层层剥离，缩小范围到模块，模块内部分割单元测试，确定优化目标。
- 4 性能优化没有尽头，在高速迭代中完成优化的同时，还需要与时俱进。移动时代的当下，不断投入和提高移动应用性能更具价值。
- 5 需要塑造情怀和氛围，追求高性能的工程师文化，写出快速友好的代码。另外，性能优化是持久战，是一个系统工程，需要耐得住寂寞。

1.3.3 相关的人

互联网企业就是产品工厂，无论移动端还是PC端，都是由产品设计工程师构图，再由软件工程师开发生产，最后由质量和运维工程师把关，以保障产品线上的稳定可用。任何一个环节都是由人及人控制的机器完成，而性能问题的根源也在于此。一定程度上，人为因素决定了性能问题的发生和修复。而往往一个产品的性能问题是由多个人导致的（很多性能问题的产生是因负责人的意识淡漠及“功力”不足而直接导致的），我们能做的是将这些人通过性能相关的数据，联动起来，一起解决并保持。

应用性能管理的角色与分工

众所周知提高一个网站的性能很难，而以一个很小的团队让一个大规模网站一直保持高性能则更难。性能问题可以发生在一个模块，也可以是一个产品，更可以发生在一个部门的多个产品。基于一家公司对所有产品进行对应性能优化，就是最大规模企业级的全公司产品性能优化。例如腾讯、阿里、百度都有专职优化团队来负责公司级产品优化。而我在百度期间负责的UAQ团队就是具有这样使命的团队。在日常工作中，需要同时配合14条核心产品线进行性能优化。而通常比较多的情况是产品级性能优化，一个产品团队的技术经理负责组织前端、后端、运维等同时立项，分期优化达成目标。通常不同角色的分工如图1-3所示。

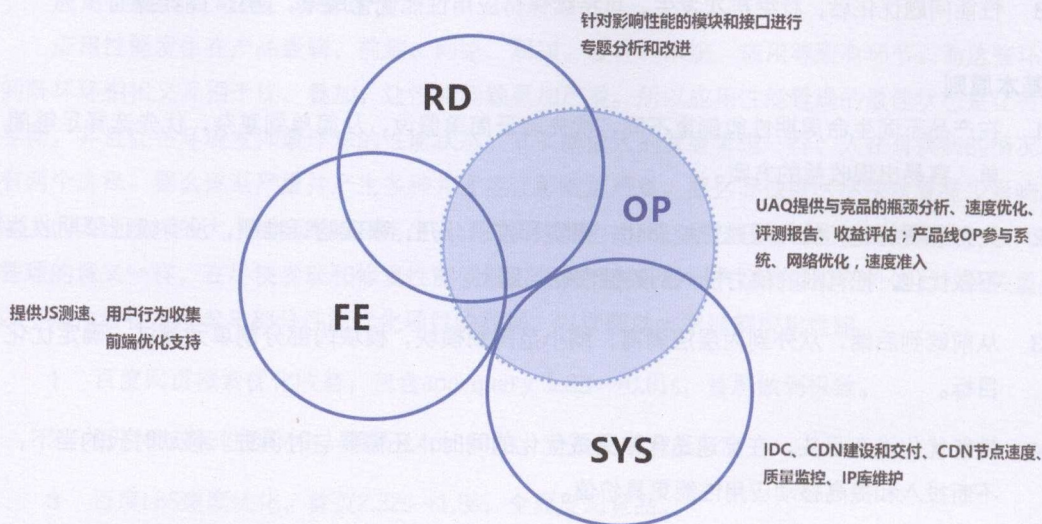


图1-3 性能优化团队角色及分工

1 FE（前端研发）

- 1) 分析前端程序执行效率，改进前端逻辑和代码性能。
- 2) 前端公共框架和代码质量管理，前端发布质量规范。
- 3) 负责前端JS测速、用户性能数据收集和分析，前端性能分析工具开发。

2 RD（架构师、后端研发、移动研发）

- 1) 分析程序执行效率、系统架构中的性能瓶颈，优化系统结构。
- 2) 设计更好的应用系统架构和落地。
- 3) 针对影响性能的模块和接口进行专题项目改进和持续迭代。

3 OP（运维）

- 1) 分析系统运行状况（系统），分析系统资源使用情况(硬件)。
- 2) 分析应用程序对资源的使用情况，应用程序执行效率及相关压力测试。
- 3) 负责服务器硬件、软件、软件配置性能优化，前沿相关技术调研和落地。
- 4) 提供与竞品的瓶颈分析、速度优化、评测报告、收益评估。

4 SYS（网络）

- 1) IDC、CDN、硬件性能测试、商务采购、上架，新硬件调研和落地。

2) 负责IDC外网、内网传输及相关QoS保障, 负责IDC、CDN性能优化。

3) 负责操作系统母盘、内核、TCP传输、网络路由等调优。

正视团队在性能管理方面的短板

性能问题发生在成长型团队或产品快速迭代的团队中是很常见的, 而发生在成熟团队中则较少见到。团队负责人是性能问题发生和优化的主要负责人, 起到性能问题的关键把控作用。通常看一款产品的性能问题, 就能看出团队负责人的底蕴。可以预见的是, 如果团队负责人能规避多数性能问题, 这款产品的体验肯定是优秀的, 并且这个团队的成员在性能优化的储备肯定是足够的, 反之则刚好相悖。从我在腾讯、百度做性能优化的这几年来看, 正常情况下, 多数团队达不到理想状态与团队负责人出现短板和没有意识是最糟糕的情况, 而团队中部分角色出现短板则是很常见的情况。

无论何种原因导致性能瓶颈, 无论是谁指出来我们的产品存在性能问题, 团队负责人一定要重视。因为只要存在性能问题, 就会影响使用产品的所有用户, 包括未来的新用户。如果不优化, 放任自流, 就会持续影响。团队负责人不仅需要具备规避性能风险的能力, 更需要提升整个团队性能优化的意识。只有在日常迭代中减少性能问题的发生, 才能从根本上保证在新产品开发过程中杜绝性能问题。值得注意的是, 在保持和优化应用性能的经验教训上, 以下三个方面也非常重要:

- 1 找到“头狼”, 让有经验的性能优化高手来主导并快速启动性能优化, 快速聚焦最高价值的优化目标的同时, 在过程中提升整个团队的性能优化能力和意识。
- 2 一切以数据说话, 优化的决策和优化后的度量都是要建立在科学、能说服所有人的性能数据之上, 并能监测最终用户体验及优化收益, 把性能数据价值化与全团队分享, 最终得到认可。
- 3 分担责任, 性能问题的发生是由产品各环节的角色决定的, 首先要自己担当并正视性能问题。性能优化的成功必须建立在与产品研发、测试、运维等团队长期合作之上, 不可能单方向完成, 更不可以一蹴而就。

1.3.4 解决的问题

所谓信息量是指从 N 个相等可能事件中选出一个事件所需要的信息度量或含量, 也就是在辨识 N 个事件中特定的一个事件过程所需要提问“是或否”的最少次数。应用性能优化亦如此。每家互联网企业都有若干产品及产品对应的开发环境、测试环境、生产环境, 以及这些环境运行的IDC、服务器、系统、应用、用户端等都同时存在大量的性能事件。这些性能事件最终会综合发生在用户使

用产品的体验中，而应用性能管理要解决的问题也应运而生。以下将结合个人实践，提出应用性能管理主要待解决问题。

最终目的是提高用户体验

用户体验与产品影响力、企业形象甚至企业营收都有直接关系。性能管理最终目标都是以提高用户体验为根本。例如BAT都是以追求极致体验著称。一切应用性能优化和资源投入都是围绕核心产品展开，力争在同行业将产品体验做到最优。百度在提升搜索体验中不断探索和追求极致，为了加快100ms，投入数千台服务器，足见其对于用户体验重要性的把握。

让应用性能完全透明可控

建立完整体系的性能监控和海量实时数据分析平台，有助于实时跟踪页面性能，及时发现页面性能问题，并为性能优化效果提供数据支持。应用性能管理最基础的工作是监测所有，宛如雷达一般的作用。也只有全方位监测到所有环境中的性能事件并将这些性能逐一可视化、趋势化，才具备应用性能管理的前提。应用性能中衡量用户体验的指标超过100个，分布在移动、PC、网络、系统、应用等大的维度，后面的章节会详细介绍如何搭建体系的监测平台并介绍如何评测相关性能指标。

积累团队性能优化实践

积累性能分析和评估方法、研究性能优化的方法，有助于提高工程师在优化性能方面的知识。而提供优化性能的工具和库，为多个产品线性能优化提供支持，则能让所有人都考虑性能，快速实施性能优化。性能问题经常在线上被用户反馈或人工巡检发觉，这在初创团队或新产品团队中是很常见的现象。一方面，因为成熟的团队或大的产品线都具备足够的人力和时间来完善应用性能，但肯定都是从不完善中走过来的。另一方面，由于团队在不断地流动和变化中，这需要团队不断积累与性能相关的意识。从一定程度上讲，应用性能对于研发抑或运维都属于高阶技能，需要在日常工作中不断实践和积累。

自动化解决常见性能问题

研发、测试、运维等工程师的主要压力源自实现产品性能稳定、可持续发展。在大多数情况下，由于主观上对性能优化方法和工具了解有限，而学习和实践又需要较长时间，所以最理想的状态是实现产品线优化工程化地解决性能问题——即能自动化地规避、优化应用性能。但是当下对多数团队来说可操作性极弱。以下列举较为常见的实践场景，以便理解。

- 1 成熟互联网企业，例如BAT有公司级统一高性能前端框架、统一CDN、BGP网络接入、高性能应用组件等相关部门，并且有专业的人来维护这些平台。产品只需要拆分接入即可享受

到最优性能，而且不需要自己维护。除此之外，成熟的互联网企业应当具备完整的应用性能管理体系和意识，在产品测试和发布前尽量将性能问题发生的概率减至最低。

- 2 逻辑简单的产品，例如以静态模块为主的电商、咨询、UGC用户产生内容等类产品，大量内容即是用户访问的对象。这些内容在发布时就可以自动做大量的优化，并自动发布。也可以使用第三方高性能的云，例如计算、存储、网络等。

1.3.5 前提条件

具备分析问题能力

具备应用性能相关的基础知识和学习能力是做应用性能管理的基本条件之一。任何企业和团队有关应用性能管理的经历都是由小到大、由浅入深的积累过程。我在腾讯工作时，腾讯成熟团队最先开展性能优化，并不断将优化成果在公司内部分享。这是我们当时最快的学习途径之一。由于部门和产品之间存在差异性，所以我通过消化其他产品团队的优化思路和方法，结合个人的体会，总结几个主要学习途径如下。

- 1 公司内部。相对大的互联网企业，多部门多产品，有些产品和团队肯定走在最前面。从研发的技术发展通道看，前端技术方向肯定会涉及性能优化。因为前端负责看得见摸得着的产品构建，直接产生大部分性能问题。而在公司内部首先找到这些部门和接口的人，安排部门之间技术交流或当面沟通都可以快速学习。
- 2 专业大会。当前是用户至上的时代，各公司或多或少都有不少性能管理的实践，在各种大会上都能看到相关的分享，这些都是学习的途径。例如Velocity大会的主题就是性能，InfoQ上也有很多与性能相关的采访和文章。
- 3 多认识牛人。无论是国内或国外公司都有这个方向的牛人。而性能优化本身是互联网技术体系里的“上乘武功”。往往这些牛人都可以通过朋友介绍或在微博、微信上找到，也可以通过各种大会认识，而且这些牛人多数都有在网站写博客等习惯，这些都是学习和交流的突破口，如果问题较多可以整理出来集中约时间或邮件交流。

争取领导和同事支持

首先可以肯定的是，所有领导在用户体验和应用性能上都是愿意投入时间和人力的，只是需要引导领导去正视现在的问题及排期。其次，每个产品团队的构成和长短各不相同。有些团队的负责人是产品出身，有些是研发出身，有些甚至是销售出身等，这些都需要通过对应的侧面去引导。此

外，还存在一种情况是领导需要更多的数据做支撑，需要说服相关的同事配合去做一些问题的验证，做出几个收益来辅助说明。最理想的状态是让领导亲自摇旗呐喊，落实充足预算，与奖金挂钩。举个例子，阿里有个产品团队就与总裁级别的领导对赌过应用性能，即设定速度KPI，如果完不成KPI当年考核最低档，当然如若完成KPI奖金很丰厚。

多了解公司内外资源

性能优化要想做得深入，需要建立一整套监测、分析、优化平台，特别是监测平台。而很多团队初期往往是从零开始的，一方面需要借助公司其他团队和公共团队的相关平台来拿到数据，才能将数据二次加工集中到自有平台。除了相对监测起点较高外，优化的资源更容易获取，例如网络、硬件可以预算采购、人力可以协调排期、公共的基础平台可以沟通接入等。另一方面，也就是公司外的资源，主要是通过找到优秀的基础云平台，并将这些云平台的技术人员也吸纳进来共同达成目标。

1.3.6 组织形式

多数互联网产品团队往往因新产品功能和迭代而忽视应用性能，但当应用性能的问题积累到临界点后，会毫不留情地以影响产品体验的方式体现在产品的使用过程中，从而对产品的总体价值产生负面影响。而要平衡这个临界点是需要具备一套完整的应用性能监测与分析平台，这些工具平台的建立需要时间和不断调优。所以优秀的企业会招聘专职的性能优化工程师来搭建监测平台和分析应用的性能，从而帮助企业多个产品团队可持续优化。这些性能优化团队与产品团队是可平行发展的，甚至可以理解为内部的甲乙双方关系，这是常见的企业级应用性能管理的组织形式。

虚拟团队

组织保障是非常有必要的。正所谓闻道有先后，术业有专攻。产品线RD\FE\QA\OP团队主要精力集中在生产和维护产品上，而性能优化团队的主要职责是在系统化的性能分析与优化上。实际情况是，性能优化工程师综合能力和技术等级越高，产品线团队与性能优化团队两者则更能完美互补。在一定时间段内，两个团队需要组成一个虚拟团队并设定一个性能优化目标。例如在腾讯做门户性能优化时，我们会持续对比三大传统门户网站的速度，优化前排名第4，经过优化后完全反超竞争对手。在百度PC搜索、移动搜索也是类似的形式，最后经过虚拟团队的协调和持续优化，最终全面反超竞品。

值得一提的是，也可将第三方相关团队纳入到虚拟团队中。例如第三方CDN、APM、TCP加速服务商等。特别是在中小企业内部人力有限的情况下，第三方的人力是非常好的补充，而且他们经验丰富，服务和配合能力较强。通过第三方团队可以了解业务，并借助其曾经帮助过的其他企业实践

过的优秀经验，直接转化为提高自身企业应用性能优化效益。

资源

古人云“巧妇难为无米之炊”。其实也有这层意思，这里说的资源主要指非人力因素，专用于支撑应用性能改善的资源。例如网络资源IDC\CDN\BGP\TDO、内部或外部高性能云平台等。这些资源之间需要提前协调，在不同的阶段要沟通好到位的时间。如果过早就绪会直接导致闲置、浪费成本。举个例子，如果在Q2需要使用100台全新服务器，而在Q1就已经全部上架，将导致90天的机架租用成本、电费及服务器折旧。

平台

这里的平台主要指应用性能的监测、分析、优化平台。其中监测权重最大，前期性能监测不到位或监测不完整都会让性能优化失真，甚至是偏离方向。后期则需要持续监测应用的性能并保持性能优化的成果。监测按大的分类主要可以分为PC、移动、国际化等，例如移动监测又细分为Web App、Native App（iOS\Android）监测，其中PC监测范畴最广。主要分为PC和移动真机监测、PC和移动JS监测、网络监测、可用性监测、流媒体监测、应用监测等。监测是应用性能管理的“眼睛”。可以说如果没有监测平台，一切性能优化都将失去度量，更无法发觉性能问题。

笔者之前在百度负责UAQ性能优化团队，结合个人体会，给出如图1-4所示的企业级性能优化组织形式。

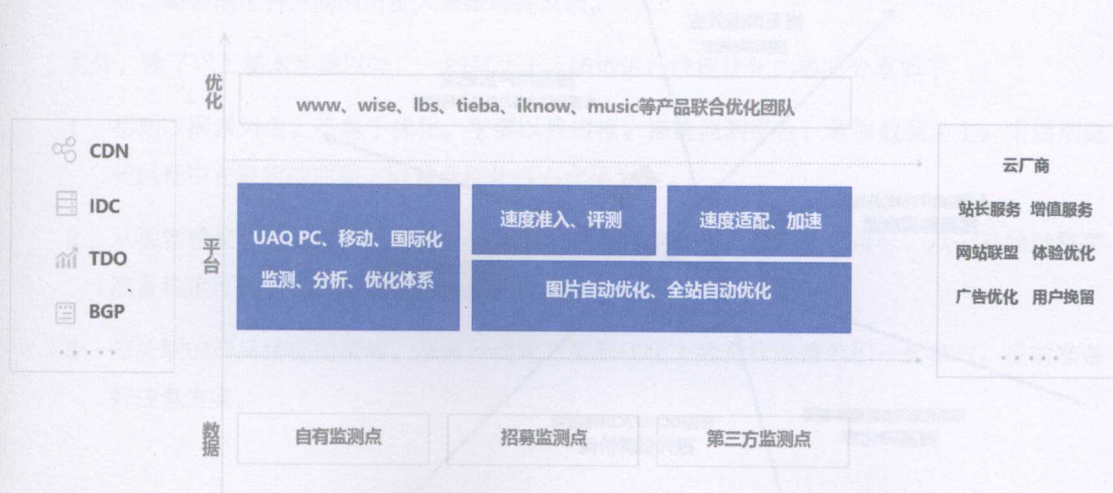


图1-4 百度优化团队组织形式

1.4 如何正确开始

应用性能管理三部曲

性能问题是实时发生的，并且会像人生病一样反复出现。对于互联网企业或产品负责人，需要持续地关注应用性能，并不断进行改进。而面临的最大问题是如何正确地开始，少走弯路。纵观国内外多个优秀企业的性能优化团队沉淀，和自己在腾讯、百度的实践，个人认为性能优化主要围绕监测、分析、优化三个核心循环。第一步，监测最是关键。监测好比掌握应用性能的“眼睛”。可尽管万事开头难，但良好的开端即是成功的一半。所以我认为前期的最主要精力应该放在监测上，监测对象、如何监测、监测数据如何分析、如何定位故障并优化等问题都会在后面的章节详细阐述。应用性能管理的三步如图1-5所示。

- 1 **监测**。通过监测产品及竞争对手在各终端、各产品形态下的性能，例如PC、手机、平板终端及操作系统、网络、应用等，全面评估自身及竞争者的表现，并迅速定位故障及排错。
- 2 **分析**。通过标准来评估网页/应用/网络IDC、ISP、CDN等性能，为优化及资源投入提供依据，为优化效果提供衡量，提供性能及故障预警、报警。
- 3 **优化**。通过网络、系统、前端、应用等各层进行体系优化，以将产品网页速度优化提升至业界最快为目标，进而提高用户忠诚度、购买意愿、品牌价值等。

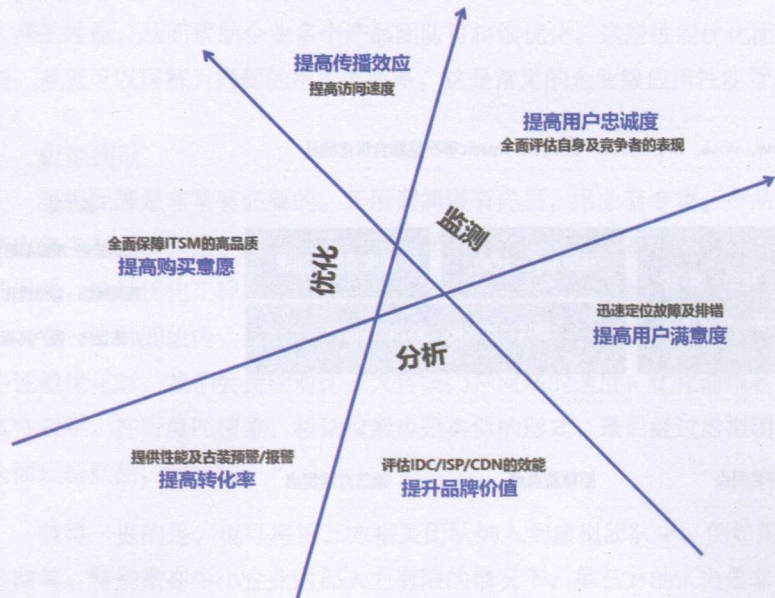


图1-5 应用性能管理的三步

如何正确开始

- 1 部署监控。前期最重要的是部署监控，无论自有监测平台还是第三方监测，必须在实际进行优化前完成。因为如果没有对比测试，就无法证明优化效果。只有通过监控平台获取到竞品及网络相关的数据，才对确定优化方案具有参考价值。
- 2 查找瓶颈。接下来对于这些监测对象和数据，需要分析出影响性能的瓶颈在哪？有哪些方面需要不断调整或深入监测？这一阶段的主要工作是搜集各种相关信息，方便接下来制定优化方案。
- 3 确定方案。完成瓶颈查找后的下一步是确定优化方案。通过分析之前收集到的各种数据来进行判断，并完成需要发起性能优化的讨论及确定人力、资源投入的统筹。
- 4 优化实施。这一步主要指优化方案的实现过程。例如研发从前端、后端、移动端进行全方位的代码和逻辑优化及硬件、网络升级等。由于某些优化会导致一些关联的问题发生，需要做好前期的沟通和充分的论证。而核心产品需要单独搭建线下测试环境，经充分测试没有问题后，在上线过程中还可以灰度做A/B测试，通过监测真实用户的数据，选择最优方案。
- 5 跟踪反馈。持续反馈也是非常重要的工作，性能优化是一个繁杂的系统工程，除了参与人和优化项多以外，有些优化是需要长时间持续优化的。并需要进一步建立周会或报告机制，需要指定各方向的负责人来跟踪并反馈。

另外，除了以上基本步骤以外，一些有助于正确地进行性能优化的思路分享如下：

- 1 初期以探索为主，不急于优化。主要以找短板，搭建监测平台，掌握数据为主。中后期迭代过程中在稳定的前提下以重点优化核心产品为主。
- 2 从视觉感受和数据监测上同时分析问题。在数据的基础上设定优化目标，从一开始就需要准备稳定可靠的长期应用性能监控机制。
- 3 与关联的产品线提前接触。从设计优化方案到优化上线全程邀请他们一起参与，提前准备好应急方案。

1.5 投入与收益平衡

做好长期投入的准备

无论应用性能管理生命周期中的监测还是优化，都需要投入资源才有产出。例如人力、服务器（包含硬件）、机架、IDC\CDN带宽等。这些资源投入只要有科学的监测数据来衡量引导，肯定会有收益，只是收益的大小和质量不同而已。事实上，往往在性能优化立项之初，我们只需知道要达到的总目标，例如80%的用户加载时间 $<2s$ 。但要达到这个目标不可能一蹴而就，需要分期分阶段完成，特别是在大型互联网企业周期更长。例如我在腾讯做门户性能优化时，一共花费2年时间历经3期；在百度做搜索性能优化时，共耗时3年。所以需要从一开始决策投入与收益，基本原则如下。

- 1 抓住主要矛盾，制定性价比最高方案、让收益最大者先行。通常立项时已经能够大致分析有多少改进空间，这些改进空间需要什么样的资源。例如使用CDN，将旧业务迁移至高性能公共平台等。需要将最容易落地、最容易出收益的先做，通常系统层、网络层的优化最容易出成果，受益面最广，而且是一劳永逸。
- 2 联合到所有相关的人并协商好时间并行。例如前端工程师、移动工程师、运维工程师可以并行，也可单独进行优化，优化上线时也可以约定好时间共同上线，合并收益。也可以分前端、后端、移动端、网络端等同时进行。总之，灵活机动，合理分配。
- 3 提前协调资源，特别是贵资源。例如IDC、服务器，至少需要提前半年做预算，并最终通过发起采购、生产、物流、上架、配置环境和应用等一系列复杂的流程才能落地使用。人的时间也是需要提前协调的，特别是部门的高工或架构师，基本全年都排满了，多数情况下需要BOSS特批才能腾挪出有限的时间。还有一种常见的做法是借，借机器、借人，都需要上层领导的足够支持。

应用性能优化 VS 成本优化

首先明确一点，性能优化最大的收益是改善用户体验，增加产品的口碑和商业价值，且同时还存在另一种极为重要的收益，即运营成本收益。结合我在腾讯、百度参与了大量的产品性能优化项目经历之见，由于系统总会有历史包袱，初期随便抓一个产品，分析后总能看到或多或少的性能问题，可随着优化工作的进展，越到后面，优化的工作慢慢变少，保持和维护优化之后的性能慢慢成为主要工作。我在众多性能优化实践中得到的经验是，应用性能管理的初期，优化空间最大、优化收益最容易体现。这里的优化空间往往还包含大量的运营成本减少，性能优化与成本优化通常在优化前期是能够两者兼得的，而优化后期优化空间和难度越来越大是需要用成本换速度的，两部分的关联如图1-6所示。

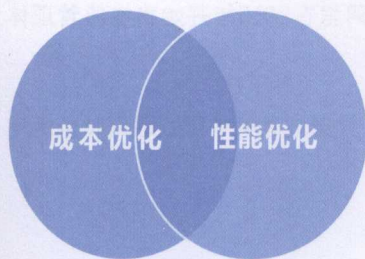


图1-6 性能与成本优化示意图

性能优化伴随大量的页面和应用性能提升，而内容瘦身、业务合并会直接降低带宽成本和服务器投入。恰逢这两种成本是互联网企业运营成本主要构成，性能优化通常伴随以下成本减少。

- 1 内容瘦身和大量使用全国CDN，减少、北上广中心IDC带宽和服务器使用，降低带宽、机架、服务器成本。
- 2 提升后端模块性能和迁移规模化的高性能集群，从而降低资源消耗，提高资源利用率，合理冗余以抵御灾难。
- 3 通过服务器性能优化和管理，让资源管理透明化，预算估计更规范化，以减少不必要的资源闲置和浪费。

1.6 优秀企业的经验

国际一流互联网企业（Google、Facebook、Yahoo! 等）

- 1 性能管理方法论的先驱，并成立独立的性能优化团队，将性能优化发挥到极致。
- 2 开源和沉淀了大量性能管理原则、工具、流程。其中，Google将性能作为网站检索的重要权重。
- 3 性能优化领域最专业的Velocity大会，每年为业界分享最新、最前沿的性能优化相关成果。
- 4 Yahoo!有专业的性能管理团队，并对外产出工具和指南，例如Yslow。
- 5 Google有专业的性能管理团队，并对外产出工具和指南，例如webpagetest/pagespeed/SPDY。
- 6 Facebook有专业的性能管理团队，建立大量应用性能数据统计和分析模型。

国内一线互联网企业（腾讯、阿里、百度等）

- 1 紧跟国际先行者，针对自身特性进行大量性能优化实践，并不断总结、完善。

- 2 腾讯从2007年就开始进行大规模用户体验优化。百度、阿里在2010年开始将性能管理体系化。
- 3 公司级性能监测、优化团队随着逐年性能优化不断深入，并形成具有中国互联网特色的完整性能管理方法论。

国内二三线互联网企业（新浪、携程、美团、58同城等）

- 1 近年开始进行性能优化实践，并取得重大收获，性能提高20%~50%。
- 2 逐渐深入到研发、测试、发布等产品生命周期，在性能上不断领先竞争对手。
- 3 性能优化相关体系日渐完善，并不断将性能优化成果在各主题大会上分享。

笔者有幸在百度负责过用户访问质量UAQ（User access quality）团队。团队定位主要是负责百度产品应用性能管理与优化，通过建立用户访问质量监测平台，提供百度产品及竞品访问质量一对一的监测、分析、优化服务，进而建立可持续的用户质量分析、评估体系和优化最佳实践。主要通过内容和应用、网络、系统层等优化，达到提升用户体验和产品价值的目标。团队工作覆盖百度33个产品线，监测1200个任务，1~2重要级的产品线覆盖率为93%，输出300+份评测报告，全力配合百度14条核心产品线性能优化，性能管理的形式如图1-7所示。

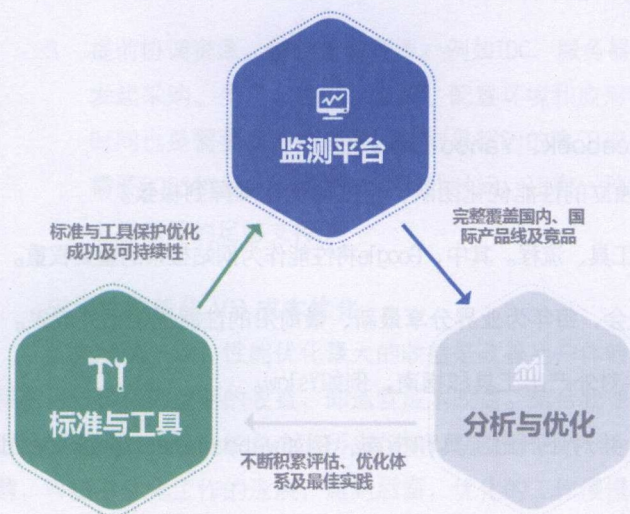


图1-7 百度应用性能管理实践

第2部分

监测、工具篇

第2章 应用性能监测实践

第3章 性能监测工具介绍

第4章 性能监测平台搭建实践

第2章 应用性能监测实践

2.1 应用性能监测概述

在现代市场上，无论是传统互联网企业，还是新型互联网企业都是由运行他们的业务软件产品定义价值，进而产生营收。也就是说产品的用户体验将直接影响企业的收入和口碑。由此可见，企业的使命不再是简单地建设一个网站或制作一个移动应用程序，然后让它们运行。而是需要一个完整、智能的监测体系进行应用性能管理，致力于时刻保持最佳用户体验。一般大型互联网企业都有专业团队或搭建专业平台来负责此项工作。

而在云时代，产品团队更迫切需要一个体系完整的应用性能监测平台，使得任何“风吹草动”都逃不出我们的手掌心。这个应用性能监测平台不仅要能够全方位监测性能数据，而且还需要具备科学、合理的数据分析能力。也就是说，应用性能监测平台会成为我们的眼、脑和手。首先它是我们的“眼睛”，帮我们7×24小时掌握整个系统的健康状况；其次它是我们的“脑”，能够根据历史的数据和策略库快速诊断并定位问题；最后它还是我们的“手”，协助我们快速进行排障和性能优化。

监测什么数据

应用性能监测平台是对应用性能数据进行分析和展现的平台。它可以用来查看页面性能的各种关键数据，并发现其中存在的问题。应用性能监测平台也通过跟踪页面性能变化的历史，了解功能升级对性能的影响，它甚至具备能够自动发现页面中出现的性能问题，并及时通知相应人员等多种功能。应用性能监测到的数据主要如表2-1所示。

表2-1 应用性能监测类型

监测数据	说明	价值	最佳监测渠道
终端用户	无论是PC还是移动产品，监测到各终端产品形态的真实用户使用体验是最有意义的。基本决定了应用性能管理的高度和成败，但实现代价和成本最高	高	PC客户端，移动SDK，JS埋点，第三方厂商
浏览器	PC、移动各浏览器及版本、各分辨率性能，用户中存在大量“小白”用户，低性能手机及浏览器用户是需要重视和区分对待的	低	PC客户端，移动SDK，JS埋点，第三方厂商
网络	PC端的宽带网络，例如网络运营商、各种本地网接入的用户是非常复杂的，而窄带移动网络及历史背景更复杂，可以说在中国不能驾驭网络的企业做不好用户体验	高	PC客户端，移动SDK，第三方厂商
服务器及应用	主要指在服务器上的操作系统及运行的各种开发语言开发的后端应用服务性能，包含关联的第三方应用，这一块是性能优化的中后期的主“战场”	高	Agent，日志，第三方厂商
操作系统	PC、移动各形态产品运行在不同操作系统上，对性能的影响也是非常紧密的，特别是低性能高负载、有恶意软件及存在劫持、操作系统被定制过的环境等	低	PC客户端，移动SDK，第三方厂商
厂商	主要指移动手机厂商，不同厂商在不同型号手机上的运行操作系统。除了定制、个性化外，APP的性能也是存在差别的，包括兼容性、稳定性等。很多PC厂商的DIY原装系统也是性能杀手	低	PC客户端，移动SDK，第三方厂商
日志	日志里除了用户到服务端及服务端用户两段路径部分网络指标没有以外，其他信息基本一应俱全，特别是分析后端的性能、全量用户的性能是非常重要的作用	高	日志，第三方厂商
竞品	知己知彼是非常重要的。同类产品，用户体验比竞品优秀本身就是一种竞争力，BAT多年都在不遗余力地追求用户体验行业第一。通过竞品分析，可以知道前端、后端、网络、架构等诸多信息，用来扬长避短	低	PC客户端，第三方厂商

应用性能监测实现步骤

- 1 **采集数据。**应用性能的发生是无时不在的，并且一旦发生就会一直存在。一旦不及时处理，多个性能问题会彼此叠加放大，甚至在整个产品的生命周期的不同阶段也会产生不同规模的性能瓶颈。所以说性能数据采集是第一步，并且数据采集不仅需要完整全面，也应当随着产品生命周期不同阶段变化而变化。例如初期只需要关注基础的网络性能、系统性能、应用性能即可。而随着产品的发展和演变，需要逐步加强，在原来的基础上对多终端真实用户性能、代码性能、数据库性能等方面的全面管理和优化。数据采集中后期，在研发和架构团队的配合下还需要采集所有基础架构及关联关系的性能数据。总而言之，在采集阶段需要最广泛地采集适合于在开发、测试、运维等生产环境中运行的一切性能数据。应用性能监测各渠道分析如表2-2所示。

表2-2 应用性能监测渠道分析表

监测渠道	PC客户端	移动SDK	服务器Agent	JS	日志
数据准确性	高	高	高	高	高
数据稳定性	较高	较高	高	高	高
是否来源真实用户	是	是	是	是	是
是否需要人为参与	否	是	是	是	是
是否产生额外性能消耗	否	是	细微	是	否
采样率	低	较高	高	高	高
成本	高	高	低	低	低
可运维性	高	高	低	低	高

- 2 **分析数据。**数据分析是指将采集到所有维度的有价值的数​​据提炼并按有利于发现问题的方式进行可视化的过程。这个过程是持续的、多样的，是承上启下最关键的一步，同时也是最有挑战的一个阶段。这个阶段的成功与否取决于团队对整体应用性能的认识和实践。也就是说，优秀成熟的团队知道需要分析哪些性能数据，进而引导监测方式的落地和深入，从而论证性能优化的方向和收益。
- 3 **形成决策。**决策是经过采集数据和分析数据之后，在变化的生产环境中自上而下地观察业务的性能影响，并且能够快速解决应用瓶颈，从而提高应用性能和用户体验。最终这些二次或历经多次性能优化过的收益将由采集的性能数据趋势变化来衡量，进而循环，致力于将产品用户体验优化到最佳状态。

应用性能监测模式

按用户触发主要分为主动与被动监测。

- 1 **主动监测。**即不依赖真实用户访问，通过抽样用户或IDC环境，主动访问或模拟访问服务，得到各维度应用性能数据。这类监测是属于较小抽样监测，失真度较大，主要用来判断基本的应用性能状态。如果我们用人的健康来帮助理解，也就是说主动监测只能拿到较粗犷的生命体征状态，例如是否生病，而监测不到生什么病。而且监测频率是人为指定的，例如可用性监测，PC真机监测等，这类监测好处是不需要对产品做额外的嵌码或产生干扰。
- 2 **被动监测。**主要指真实用户在使用产品过程中产生的应用性能数据，并触发监测采集规则，将数据返回数据中心，这类数据最真实，而且抽样率可以人为控制，多少都可以指定，真实用户数据抽样率越大，数据越接近真实用户体验。例如移动SDK监测，所有JS类监测都是被动监测，往往这类监测会需要对产品做额外的嵌码并产生少量干扰。从日志中分析用户性能也属于被动监测，日志分析受日志规模和分析能力影响，时效性较难保障。

平台需要开放性和可扩展

即使有能力搭建公司级或部门级专业应用性能平台的团队，也无法同时满足公司所有产品和团队的需求，包括第三方厂商。事实上，时间和精力也不允许。例如游戏与社区产品的属性是完全不同的，电商与资讯行业侧重也是不一样的。所以要将我们搭建的应用性能平台最大化收益，让更多的业务和人使用，就必须具备开放性。开放的过程中自然需要稳定和可扩展，方便其他业务团队和研发从平台中提取对他们自己业务系统有意义的数据并植入到自己团队的业务平台，通常需要将采集、分析、告警等模块做成通用API，可以随时、无门槛调用，随时使用。

2.2 应用性能持续监测

产品的生命周期是随时变化的，而且往往会越来越复杂。结合本人在腾讯、百度多年工作经历来看，工作过的部门产品均没有小于100个，而每个产品对应的服务器数百到数千不等，对应的模块就更多了。例如百度单监控的对象已经超过了10万，每天监测到的原始日志数据达到数百TB，而且这些产品大部分都是成长型产品，只有部分产品上线跨度大的慢慢趋于稳定。无论研发还是运维都需要通过贯穿产品的不同阶段、不同层次、不同维度建立一整套应用性能管理平台，才能驾驭复杂、庞大的多产品业务系统，使其能快速、稳定地运行并保持最佳用户体验。

持续监测的基本特性

往往这样的应用性能管理平台是持续性的，并且覆盖整个产品成长过程及所有环节，伴随产品的迭代不断完善。还可以肯定的是，每一家企业因业务特性、企业文化和团队基因不同，对应用性能管理的追求是不一样的，但基本框架都类似：主要以持续监测为主，重点以用户客户端性能为核心的移动、PC监测体系及以服务端为核心的系统、网络、应用等监测体系。总体要求是监测产品所有应用场景性能，无论是互联网、移动各产品形态，还是各终端、应用、操作系统及各种网络下的性能。主要满足以下几个特性：

- 1 覆盖PC、移动、网络、系统、应用最完整的应用性能管理，适用于应用研发、测试和交付运维的整个生命周期，并支持SaaS化。
- 2 主动、被动监测最终用户应用性能及可用性。深入了解真实用户的行为，以便快速而有针对性地解决问题，并最终提升用户体验。
- 3 提高可见性就等于改进应用，提供数十种移动、操作系统、应用、公有云等应用性能解决方案，统一视图帮助一览整体性能状态。
- 4 定制和交付最终用户视角衡量业务应用性能监测体系，所有维度数据可以集成汇聚到企业系统，洞悉用户性能全局。

持续监测分类

应用性能监测主要以持续监测为主，覆盖范围最广，周期最长。7×24小时不间断监测产品运行中的所有环节性能。持续监测的数据具有趋势性，可以跟踪到长期的性能数据，有利于对应用性能进行版本迭代后的性能变化和优化后的收益评估等。持续监测覆盖所有监测维度，详细分类如图2-1所示，主要有以下几类：

- 1 移动监测，移动应用性能监测主要分为Native App SDK监测（iOS、Android）、Web App JS监测、移动真机监测三类。移动真机监测只有大的互联网企业才会搭建，即在全国分布的真实手机上持续监测移动应用的性能。不过，目前只有真机监测才可以监测竞争对手的性能，其他监测都做不到，详细内容会在后面章节介绍。
- 2 PC监测，与移动应用对应，泛指所有PC产品相关监测，主要分为PC Web真机监测（网页、文件）、PC JS监测、网络监测、真机流媒体监测、JS流媒体监测、可用性监测等，以上类型PC监测所有数据均来自产品真实用户。
- 3 系统监测，服务器及操作系统性能监测，这里要强调的是系统监测一定要秒级，这样才可以看到细微的系统级性能数据。系统监测可以通过每台服务器的监测数据汇总IDC带宽、业务模块访问量、各模块之间的上下游数据流等数据，使用场景主要是为自有服务器、自建私有云及公有云环境，操作系统主要为CentOS、RedHat、Ubuntu、Windows等。
- 4 应用监测，主要分为语言类和第三方平台类，语言类应用监测主要有Java、.NET、PHP、Node.js、Ruby、Python等监测。第三方平台类主要为MySQL、Redis、Tomcat、Docker等。



图2-1 持续监测分类

2.2.1 移动监测

移动互联网大潮势不可挡。过去这两年，如果用一个关键词来形容的话，那就是移动。它在改变一切，改变所有用户的行为，改变技术，改变商业模式，改变众多企业运营的方式。在“互联

网+"的国策下,企业的商业价值在不断往互联网迁移,而互联网的逐步演变也促使相关硬件、软件、技术不断更新。互联网向移动互联网转变已经不是趋势,而是既成事实。移动互联网相比传统互联网更为复杂,加上网络接入、手机厂商、移动相关技术局限等众多因素导致移动互联网的应用比互联网更不稳定,时常出现连接超时、闪退、卡顿、崩溃等问题,再加上移动用户比互联网用户更加缺乏耐心,所以掌握移动产品应用性能已经成为产品、研发的重要工作,是移动互联网时代做好用户体验的重要一环。

移动监测分类及对比

掌握移动用户体验第一步是需要监测到用户的真实数据,目前移动监测主要分为以下三类:

- 1 移动Web App监测, Web App运行于网络 and 标准浏览器上, 主要基于网页技术开发实现特定功能的应用。可以理解为就是一个针对iPhone、Android优化后的触屏版的Web网站。无须安装, 依靠移动设备的浏览器来访问, 前端主要通过HTML或HTML5、CSS3、JavaScript等Web技术开发, 服务端主要通过Java、PHP等实现。移动Web App监测最好的方式是在页面嵌入JS监测代码来实现, 可以指定抽样比例, 当移动用户访问应用时, 触发JS监测代码并将加载过程发送回服务端。
- 2 移动Native App监测, Native App即原生应用, 即我们通常所说的移动客户端, 是针对不同手机系统单独开发的本地应用, 如需使用则要将其先行下载至手机并安装。下载Native App的最常见方法是访问应用程序商店, 如苹果的App Store、安卓市场、Google Play等。在技术实现上一般采用针对操作系统的特定语言进行编写, 如使用Objective-C开发iOS应用, 使用Java+Android开发Android应用。Native App因为位于平台层上方, 向下访问和兼容的能力会比较好, 同时支持在线或离线的消息推送或本地资源访问, 在打开速度和交互界面上更优于Web App。因为完全基于手机操作系统, 所以内嵌移动SDK进行性能监测是最好的方式, 这种SDK与Native App一样也是基于原生操作系统, 数据最完整最真实。
- 3 移动端到端真机监测, 无论是Web App还是Native App都只能监测自身产品的应用性能, 而不能监测竞争对手的。所以如果需要监测竞争对手的应用性能, 就需要在全国多地招募真实手机或通过第三方Native App来实现移动端到端的真机监测。这类监测主要针对Web App, 因为Web App通过本地浏览器加载, 相比Native App而言非常轻, 数据也很稳定。而Native App需要安装, 如果没有像浏览器这样的公共渲染载体, 能监测到的相对公平的性能指标非常有限, 并且较基础。目前移动真机监测只有BAT和一线互联网企业才会有需求。我当时在百度负责搭建国内最大规模的移动真机监测, 将会在后面的章节中详细介绍。

另外, 常见的移动应用还有一类叫Hybrid App, 又叫混合应用, 是一种介于Native App、Web App之间的App, 它虽然看上去是一个Native App, 但只是一个UI WebView, 里面访问的是一个Web

App。Hybrid App实质是伪造一个浏览器的apk/ipa原生程序，并运行了一个Web APP。Hybrid App兼具“Native App良好用户交互体验的优势”和“Web App跨平台开发的优势”，这类应用需要看使用JS或SDK监测或两者都使用，详细对比如表2-3所示。

表2-3 移动产品形态及优劣分析

维度	Web App	Native App	Hybrid App
开发语言	网页语言为HTML5+JS	原生语言为Objective-C、Java、.Net等	网页或原生语言
跨平台性	高	低	高
开发难度	低	高	低
交互体验	差	好	较好
用户体验	干扰大	干扰少	干扰较大
性能监测	内嵌JS，性能指标来源浏览器，性能指标与网页一致，只有高版本浏览器才可以拿到完整的网络层数据	内嵌SDK，依赖原生操作系统的API，数据主要以闪退、卡顿、崩溃等Native App特有的现象	内嵌JS或SDK
竞品监测	移动真机，稳定	移动真机，不稳定	移动真机，较稳定

移动监测基本目标

- 1 为各产品团队提供针对移动应用，移动Web App、Native App、轻应用、移动API等的自动评测、监测、加速服务，帮助产品线解决应用上线后性能问题的监控与管理。
- 2 采用主动和被动不同监测模式，获取真实用户访问体验。通过实时、多维度分析和展现性能数据，及时帮助企业发现移动应用使用过程中的崩溃、超时等问题。提前预防用户流失，降低移动应用上线后维护与迭代成本，提高用户黏合度。
- 3 真实采集APP应用性能，适应于 Android 和 iOS 系统。监测 Android、iOS 应用执行的每一行代码，以及所有的请求，从响应时间、吞吐量、网络故障率、HTTP 错误率等指标展示APP应用性能。
- 4 提供丰富的视图展示，如拓扑、全国/省份/城市视图、HTTP、交互、ISP、网络、区间、错误、崩溃、设备、版本、操作系统等多种维度视图，实时发现在用户使用过程中的崩溃、慢交互等性能瓶颈。

2.2.1.1 移动Web App监测

2015年国内移动Web开发需求大增，主要受益于微信的快速发展，包括用户的增长及公众号自媒体在营销领域的渗透。2015年年底，微信的用户量达到6亿，公众号数量突破1000万。也就是说，在未来几年内，我们可以预见中小企业将大量推进品牌公众号运营，外企、政府机构将加速进驻，同时自媒体创业更在蓬勃发展。微信官方在2016年年初发布“网页开发者工具”，显示出进一步提高

微信Web内容技术含量的意图。这些趋势必然会吸引更多开发者进入Web开发领域。在技术社区中，Google依然在继续推广以Chrome为容器的Web App开发方法。从搜索趋势看，2015年移动Web App、混合式App依然是最受关注的热门技术选项。Web App监测与优化将越来越被企业和开发者重视。

Web App监测价值及目标

- 1 建立移动Web App真实用户性能监测体系，基于JavaScript，从浏览器、内嵌浏览器，收集真实用户使用移动应用时的性能数据，进而分析真实用户体验，加快对性能问题快速发现、定位及性能优化的能力。
- 2 丰富的数据分析功能，有效分析不同地域、网络、操作系统、浏览器、厂商上用户使用应用的体验，并且支持任务间的比对分析，协助灰度发布和测试等。
- 3 代码级问题定位，在JS错误、慢页面、资源加载失败等问题跟踪时，提供包括脚本文件、行数、终端环境等丰富信息，帮助精确定位和解决问题。
- 4 监测产品环境下使用的第三方API和资源性能，通过在浏览器、内嵌浏览器采集数据，既能监测本地API、服务器端API调用状况，也能监测第三方API和资源服务质量，如流量分析工具、CDN、广告等，帮助及时发现第三方问题，保证产品可用性。
- 5 轻量级快速覆盖多个Web App产品线，充分利用各Web平台的兼容性，用最小的工程代价，快速实现最大范围和类型的应用入口的覆盖。既支持普通移动端浏览器打开的网页应用，也支持内嵌浏览器打开的网页应用（如Cordova封装的本地网页）。
- 6 监测端的可扩展性，监测端支持改写指标采集算法，用于修正特定应用实现（如加载进度指示等）的真实指标数值，并支持自定义指标。

Web App监测实现原理

JavaScript监测脚本利用了多种现代浏览器的新特性，实现更高效率和更精确的指标采集，同时保留了对流行的老版本浏览器的最大限度兼容和覆盖，实现原理如下。

- 1 监测端的页面核心性能指标收集，利用了W3C正式发布的Navigation Timing规范，达到了最精确的等级。这个规范已经被所有的现代浏览器实现，能够在所有智能手机浏览器中使用。通过JS监测可以得到很多性能指标，需要理解这些指标项的真正含义，从而分析问题所在，并做出有效的优化。Navigation Timing记录的所有浏览器事件和时序，被总结成一张顺序图，清晰地展示了浏览器打开一个网页的大部分细节，如图2-2所示。对于API的详细使用方法和各个指标的细节，在此不再赘述。

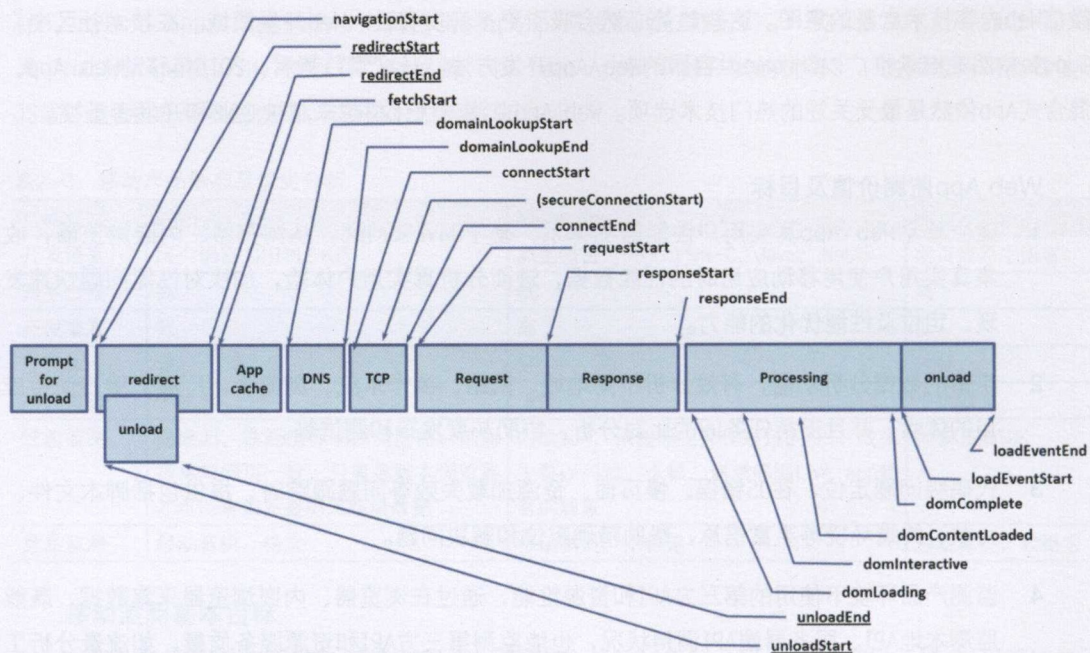


图2-2 Navigation Timing监测指标

具体的含义如表2-4所示。

表2-4 Navigation Timing指标及含义

指标	含义
navigationStart	准备加载页面的起始时间
unloadEventStart	如果前一个文档和当前文档同源，返回前一个文档开始unload的时间
unloadEventEnd	如果前一个文档和当前文档同源，返回前一个文档unload结束的时间
redirectStart	如果有重定向，这里是重定向开始的时间
redirectEnd	如果有重定向，这里是重定向结束的时间
fetchStart	开始检查缓存或开始获取资源的时间
domainLookupStart	开始进行DNS查询的时间
domainLookupEnd	DNS查询结束的时间
connectStart	开始建立连接请求资源的时间
connectEnd	建立连接成功的时间
secureConnectionStart	如果是HTTPS请求，返回SSL握手的时间
requestStart	开始请求文档时间（包括从服务器、本地缓存请求）
responseStart	接收到第一个字节的时间
responseEnd	接收到最后一个字节的时间
domLoading	loading的时间（这个时候还没有开始解析文档）
domInteractive	文档解析结束的时间

续表

指标	含义
domContentLoadedEventStart	DOMContentLoaded事件开始的时间
domContentLoadedEventEnd	DOMContentLoaded事件结束的时间
domComplete	complete的时间
loadEventStart	触发onload事件的时间
loadEventEnd	onload事件结束的时间

- 浏览器的JavaScript API监测，现代Web应用采用了大量新的浏览器API，比如说Ajax请求。“监测端”利用了较新的浏览器中Object.defineProperty接口，实现对大多数浏览器API调用的代理和监控。这样既不影响应用代码的正常运行，同时不需要开发者为了监测额外进行适配。
- 传统的时间戳收集和回传方法，在使用现代浏览器的新特性同时，监测JavaScript支持传统实现中的时间戳收集方式，在“首字节时间”、“白屏时间”等指标上达到了最大兼容性和覆盖面。并且通过对图片元素注册事件监听，安全有效地实现“首屏时间”等指标收集。在数据回传上也根据浏览器环境使用多种回传接口，确保数据在各种条件下完整高速回传。
- 配置下发模块使用CDN加速，可以允许任务管理模块动态地更新监测配置，并通过缓存刷新、设置过期时间等内部API，以分钟级速度下发大流量的配置。

Web App监测拓扑如图2-3所示。

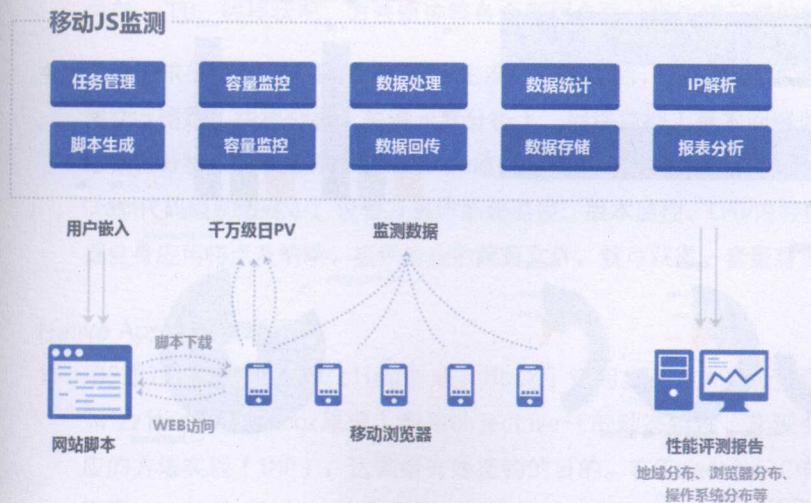


图2-3 移动Web App监测拓扑图

Web App监测视图

Web App监测核心性能指标有白屏时间、首屏时间、整页时间、用户可操作时间、首包时间、DNS时间、连接建立时间、基础页面下载时间、自定义的时间等。以上性能指标通过全国、趋势、省份、城市、运营商、区间、浏览器、操作系统等视图可视化后,提供在线性能分析功能,效果如图2-4所示,详细的视图和讲解说明会在本书第5章中介绍。



图2-4 移动Web App监测视图

2.2.1.2 移动Native App监测

移动应用诞生于苹果的应用商店App Store,在2008年7月上线之初,APP数量只有500多个,同年10月谷歌上线的Android Market仅有40多个。而截至2015年7月,前者的APP数量达到150万,截至2015年1月,后者则已突破143万。根据应用数据追踪公司AppFigures的2015最新统计显示,2014年,App Store开发者总数约为28.2万,谷歌Play Store的开发者总数为38.8万。而根据《中国移动互联网发展报告(2015)》统计显示,中国移动应用开发者数量超过300万人。从APP数量和开发者人数不断增长来看,Native App依旧是移动产品的主要形态,所以针对Native App监测与优化显得尤为重要。

Native App监测价值及目标

- 1 建立移动Native App真实用户性能监测体系,直接从终端手机提取最原始的诊断数据,采集移动APP的各项性能指标,从响应时间、网络请求、数据库、错误、崩溃、设备、CPU、内存等各个角度进行全面体检,智能地分析数据,对应用给出全面的评价,并给出完善建议。用户根据评价提示可追溯到详细信息,快速定位到问题的根源。
- 2 对HTTP请求提供详细的数据分析,全面了解主机网络性能情况,帮助产品线定位网络性能问题,提升App在不同手机版本和操作系统下的响应速度,改善App整体应用性能。对移动应用性能问题防患于未然,提高用户留存率。
- 3 提供开放性API,允许添加自定义方法监测、设置监测级别和日志级别。根据API添加自定义的方法监测,实现关键元素监测分析,从自定义方法的响应时间、自定义方法执行时的内存、CPU、线程状况、方法轨迹等各个层面全面分析关键元素的性能状况。
- 4 个性需求的定制化,监测SDK整合主流的监测功能,包括交互性能分析(基本交互分析、慢交互追踪、线程分析、关键元素分析)、网络监控(基本网络监控、深层网络追踪、网络错误分析、错误追踪)、WebView监控、多媒体监控、Crash分析(App Crash基本分析,Crash代码级别追踪)、设备及操作系统监控、版本监控、CPU/内存性能监控,等等。可以根据自身应用特点及需求,提供对应的配置文件,就可获得一套量身定制的移动监测数据。

Native App监测实现原理

- 1 iOS通过Method Swizzling方式,Hook了常用框架的API而实现了监测功能。Method Swizzling原理(Hook原理)利用Objective-C的动态特性,实现在运行时偷换selector对应的方法实现(IMP),达到给方法挂钩的目的。在Objective-C中,每个类都有一个方法列表,存放着selector的名字和方法实现(IMP)的映射关系。调用一个方法,其实是调

用selector所对应的实现方法（IMP）。因此，可以利用method_exchangeImplementations来交换2个方法中的IMP，利用 class_replaceMethod 来修改类，利用 method_setImplementation 来直接设置某个方法的IMP，从而达到Hook的目的。

2 Android通过Instrumentation代理及InvocationHandler动态代理类，使用ASM字节码框架在字节码层面Hook了android.jar中的类而实现监测功能。

1) Instrumentation代理，在Android SDK中，Instrumentation代理涉及两方面，一方面是Instrumentation代理指定了agentmain作为入口方法，实现了在main方法之前执行代理。另一方面是Instrumentation通过addTransformer方法加载ClassFileTransformer实现类，实现了在class被装载到JVM之前将class字节码转换掉，从而达到动态注入代码的目的。

2) InvocationHandler动态代理，InvocationHandler实际上是拦截器的接口。在Android SDK中，InvocationHandler代理通过其实现方法InvocationDispatcher触发invoke方法，在invoke方法中使用ASM字节码框架来分别Hook android.jar中的annotation，activity，AsyncTask，network，sqlite，exception等类方法。

3 支持移动网络、流媒体、崩溃、交互等性能监测。

1) 网络性能深度追踪，移动SDK深入系统底层，监听了C/C++层的网络通信API，获取了网络请求的DNS、TCP、SSL、数据发送、服务器响应、数据接收等各个步骤的网络交互性能数据以及详细的网络错误信息，帮助用户快速定位网络性能问题。

2) 视频、音频播放性能监测，移动SDK监测通过监听常用视频、音频播放框架，获取视频、音频播放性能数据，包括开始播放时间、首次缓冲时间、缓冲次数、缓冲总时长、卡顿次数、总等待时间、下载速度、网络状况等，帮助用户实时了解终端的播放性能情况。

3) 崩溃故障代码级分析，移动SDK不仅能监测到语言层面的崩溃，还要监测操作系统层面的崩溃信号，崩溃问题定位到代码行，历史轨迹追踪。iOS SDK利用PLCrashReporter开源框架，通过UncaughtExceptionHandler实现了对Objective-C层的崩溃信息监听，通过监听"SIGABRT"、"SIGBUS"、"SIGFPE"、"SIGILL"、"SIGSEGV"、"SIGTRAP"、"SIGTERM"、"SIGKILL"信号，捕获unix层崩溃信息。Android通过UncaughtExceptionHandler及UncaughtSignalHandler捕获了Java层及C++层的崩溃信息。

4) 交互性能跟踪分析，移动SDK通过Hook移动框架的交互API，实时跟踪APP运行时的交互

性能。帮助用户了解APP在终端设备使用过程中的卡顿、反应慢、挂起等问题。移动SDK提供慢交互追踪功能，根据method堆栈轨迹追踪，分析每一个交互的method堆栈的响应时间占比，帮助用户快速定位影响慢交互的主要因素。

4 支持移动主流开发模式及平台。

- 1) Native APP，从页面加载、页面渲染、页面Trace、吞吐量等多个角度，度量Native页面的交互性能。
- 2) Web APP，针对不同URL建立页面Trace，记录TCP、DNS、DOM加载、渲染等时间、服务器加载时间、网络状况等。从页面加载、Ajax加载、页面Trace等角度，度量HTML5 页面的交互性能。
- 3) Hybird APP（混合型APP），Hybird APP是 Native APP和Web APP的混合模式，一般Hybird APP采用原生语言的框架，对于内容经常变化、结构简单的页面则用HTML5实现。移动SDK对于Native部分，按照Native APP的监测方式监测，HTML5部分，按照Web APP的监测方式进行监测，在前端视图中分别展示Native交互视图和WebView交互视图。
- 4) 支持主流的开发平台，Android SDK需要支持了Eclipse，Ant，Maven，Android studio 四大主流开发平台，iOS SDK支持Xcode开发平台。

Native App监测拓扑如图2-5所示。

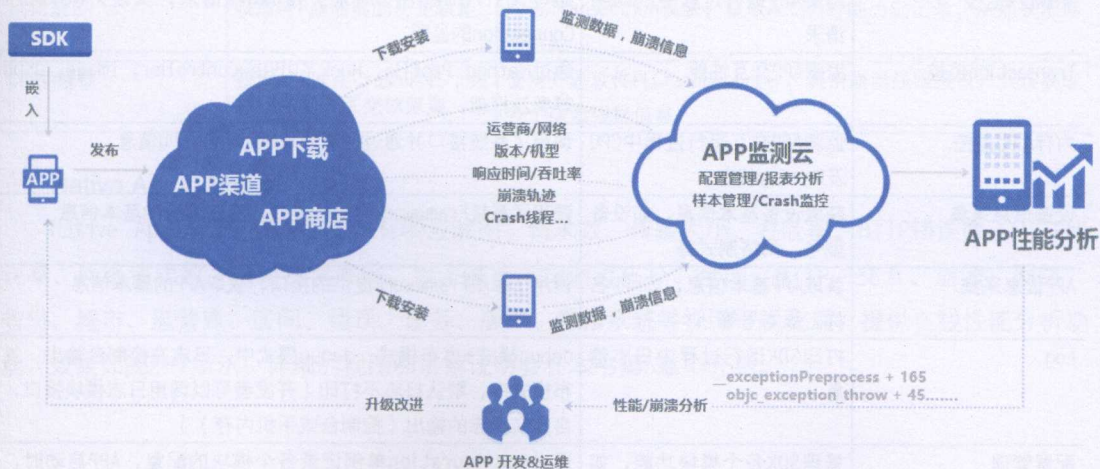


图2-5 移动Native App监测拓扑图

Native App监测架构及说明如图2-6和表2-5所示。

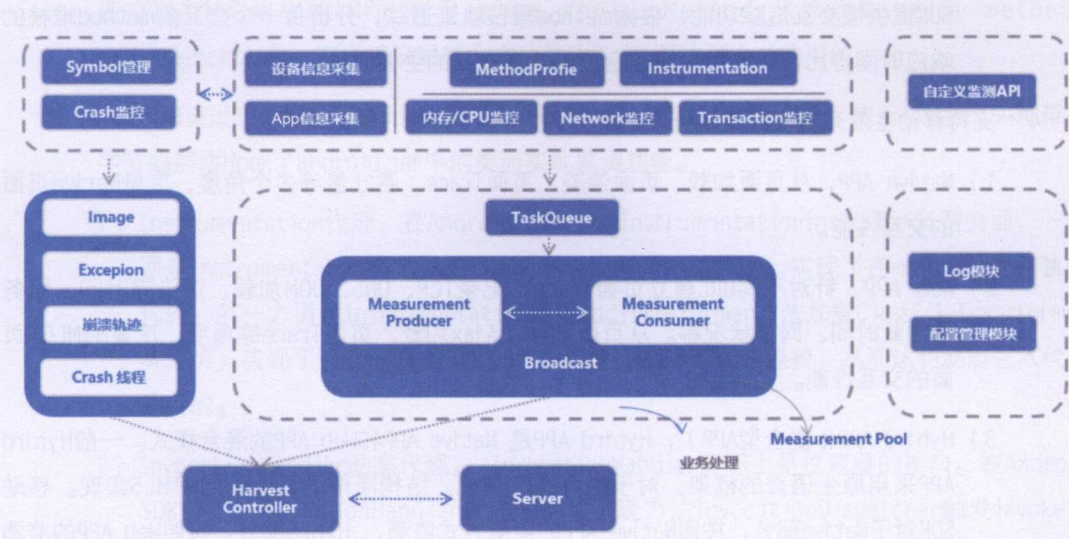


图2-6 移动Native App监测架构图

表2-5 移动Native App监测模块及功能

模块	作用	实现原理
Crash监控	监控APP发生Crash，获取Crash发生信号	使用开源软件PLCrashReport的Crash信号获取机制，实现Crash信息采集
Network监控	监测APP运行过程中的网络请求	通过Instrument，Hook了Network请求，实现对Network Connection的监听
Transaction监控	监测APP交互性能	通过Method Profile，Hook了UIViewController，Image，JSON等类及框架，实现对交互方法的跟踪
内存/CPU监控	监测APP交互运行过程中CPU及Memory状态	调用了系统接口并通过计算获取Memory、CPU信息
设备信息采集	获取设备基本信息，如设备型号、网络制式等	调用了系统Framework提供的接口，获取设备的基本信息
APP信息采集	获取APP基本信息，如APP名称、版本号等	调用了系统Framework提供的接口，获取APP的基本信息
Log	打印SDK运行过程中日志信息	debug模式+发布模式：dedug模式中，日志在控制台输出；发布模式中，默认日志不打印（开发者可以调用日志模块接口，自定义日志的输出（控制台或手机内存））
配置管理	管理SDK各个模块功能，如功能模块的开关等	通过Configuration单例记录各个模块的配置，APP启动时，Configuration为默认配置，通过与后台交互，可返回个性化配置
自定义监测API	开发者可通过API自定义监测方法	根据API参数@selector，通过反射，实现对特定方法的跟踪

续表

模块	作用	实现原理
HarvestController	周期性将采集的数据发送给Server	Timer + Connection, 通过定时器周期性地将采集的数据发送给Server
MethodProfile	对Transaction模块各个监测方法 Hook	根据Method Swizzling+runtime机制, 实现了对Transaction模块各个监测方法Hook (此模块具有良好的可扩展性, 后期添加监测方法, 只需添加类名及方法名)
Instrumentation	对Network模块, 各个监测方法的Hook	根据Method Swizzling+runtime机制, 实现了对Network模块各个监测方法 Hook
Symbol管理	Symbol文件用于将Crash日志中的内存地址符号化	通过Python脚本, 将每一个版本的Symbol文件发送给服务端
TaskQueue	临时存储各类数据模型, 并分发任务	通过单例+list, 暂存各个数据模型, 通过Schedule将任务分发给Measurements做数据处理
MeasurementPool	各类Measurement存放池	存放各类Measurement实体类, 并通过broadcast, 供Producer和Consumer转化
Measurement Producer	将各个实体类转换成Measurement	通过实现各个实体类相对应的producer, 将各个实体类转换成对应类型的Measurement
Measurement Consumer	将各个Measurement做处理, 转换成Harvestable数据	通过实现对与Measurement类型相对应的Consumer, 将Measurement转换成Harvestable数据
Image	Framework名称和内存地址对照, 用于符号化内存地址	使用开源软件PLCrashReporter, 调用获取当前Image信息接口, 实现获得Image文件
Exception	获取崩溃异常信息	通过ExceptionHandler获取Exception信号, 从而调用Exception框架接口, 获取Exception信息
崩溃轨迹	获取APP崩溃前的历史轨迹	通过Transaction模块, 获取Activity的历史记录, 从而获取崩溃轨迹
Crash线程	获取APP发生Crash时各个线程的状态	使用开源软件PLCrashReporter, 调用崩溃线程接口, 实现获取各个崩溃线程信息

Native App监测视图

Native App监测核心性能指标有响应时间、请求数、传输大小、吞吐率、HTTP错误数、HTTP错误率、网络错误数、网络错误率等。以上性能指标通过拓扑、全国、HTTP、交互、崩溃、网络、省份、城市、运营商、区间、错误、设备、版本、操作系统等视图可视化后, 提供在线性能分析功能, 效果如图2-7所示, 详细的视图和讲解说明会在本书第5章中介绍。

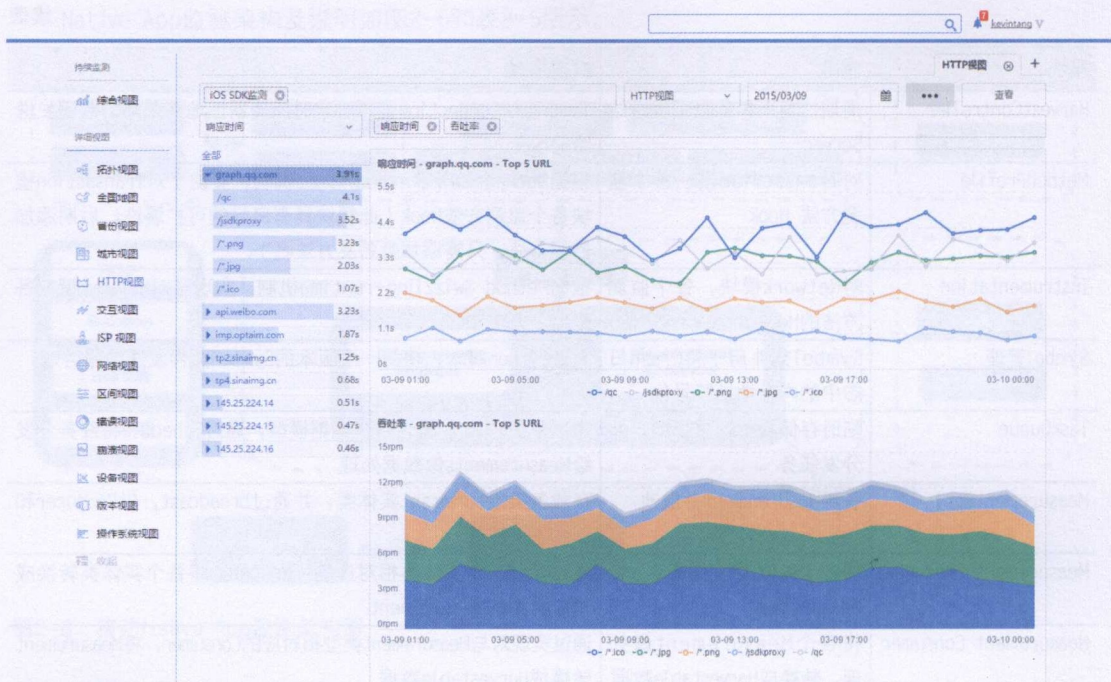


图2-7 移动Native App视图

2.2.1.3 移动端到端真机监测

随着移动互联网持续发展和深入，可以预见无论是现在还是将来，移动APP依旧是企业主要商业价值载体。而在商业竞争中，“知彼”远比“知己”重要。不过无论是移动JS监测还是移动SDK监测都不能监测竞争对手性能，所以企业对Web App、Native App产品形态的竞争对手性能监测的需求随移动互联网演进而越发强烈，移动端到端监测应运而生。

移动端到端监测的所有监测过程发生在真实手机上，监测手机小规模数量可以部署在测试环境（多厂商、多网络制式），而监测手机大规模数量则主要通过自有移动客户端下发或有偿招募。百度性能优化团队建立了一整套移动真机监测平台，主要对百度移动搜索等核心移动产品的竞品进行性能监测。招募真实手机超过3万台，覆盖33个省和港澳台地区264个城市，同时在线1500+，日活5000+，采集样本数10万+/天，从用户、地域、厂商、运营商等维度建立了详细的移动竞品性能监测和分析体系。另外，Web App竞品监测因监测过程发生在浏览器，相对Native App要简单。移动端到端监测拓扑图如2-8所示。



图2-8 移动端到端监测拓扑图

2.2.2 Web监测

在“互联网+”的浪潮之下，不仅在互联网本身这个领域产生了非常大的影响，其实也渗透到整个国民经济的方方面面，甚至可以毫不夸张地说，对各个行业都产生了非常大的影响。现如今，中国互联网发展进入了第二个十年，期间积累了大量的互联网产品形态。这些不同互联网领域经过梳理划分为资讯、搜索、娱乐、游戏、电商等行业。这些看似纷繁的各行各业，除了都经历过在互联网不同阶段中漫长时间的不断演变以外，还具备一个共同点，它们都属于PC时代。所以，这里讨论的Web监测主要指传统PC互联网及网络相关的范畴，可以参考图2-9帮助理解。Web监测是每一个互联网企业都会面临的问题。

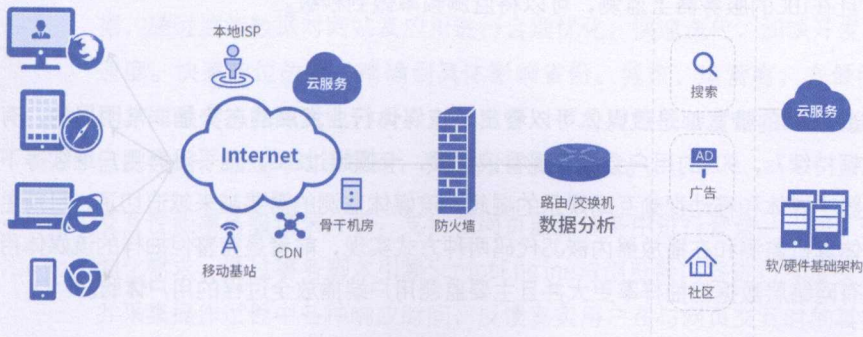


图2-9 Web监测拓扑图

PC端到端真机监测

通过招募真实用户安装PC监测客户端或企业自有PC产品客户端进行应用性能监测，监测对象主

要为网站/网页/网页元素/事物流、应用API、IDC/CDN、服务器/云主机等。以抽样最终用户的视角及时发现在访问过程中遇到的性能问题，通过对应用性能及监测数据不断的优化，使最终用户获得更优的用户体验。PC端到端真机监测是目前唯一可以监测竞品的监测途径，而且是主动式、无须添加任何代码，目前市场上第三方这类监测厂商主要有国外的Dynatrace的前身Compuware、Keynote及国内的基调、博睿。值得说明的是，目前国内这种监测也是APM行业主要的变现途径。

PC JS监测

前端嵌JS收集性能数据是零成本的，代价是通过前端工程师在网页中嵌入JS监测代码，并会产生100~200ms左右的延时消耗，正如监测方式一样，这类监测是偏网页和前端的性能监测，主要由前端工程师主导，而且自定义非常强，几乎想监测网页中的任何元素都可以，例如白屏、首屏、全屏或网页中的某一块区域加载性能等，因为需要人为添加代码，监测效率较低。

网络监测

无论是传统互联网还是现代云时代，所有应用和企业商业价值都依托在网络之上，况且中国基础网络是非常特殊的，简单来说就是有天然屏障。可以很肯定地说在中国能驾驭基础网络的互联网企业用户体验一定不会差。网络监测更是每家互联网企业都需要的。目前主流的且接近真实的网络监测主要是PC端到端真机监测网络延时和内嵌JS下载不同IDC中的服务器上的文件判断网络性能。

可用性监测

可用性监测有一定的历史渊源，也是发展最早的一类监测，主要通过一个或多个IDC向服务器、网页或API发送请求，通过返回的数据判断应用的状态，通常这种状态只有可用与不可用两种，因为这类监测没有任何真实用户体验的意义，只局限于辅助快速发现故障从而提升用户性能，只能判断应用的可用状态，好比去敲门，如果房间里有人应答应说明房间里有人，是单纯的可用性监测，加上实现简单而且在IDC的服务器上监测，可以将监测频率做到秒级。

流媒体监测

从主流CDN厂商超过50%的带宽都是流媒体可以看出，流媒体行业发展的态势是非常明显的，有权威数据表示视频卡顿持续7s，90%的用户会放弃观看该视频，卡顿超过20s，几乎没有用户继续等下去。加上近几年家庭数字媒体和移动视频互动娱乐的崛起，流媒体监测的需要越来越迫切了，目前主要通过PC端到端流媒体真机监测和在播放器内嵌JS代码两种方式实现，前者是完整但抽样的流媒体用户体验监测，后者没有网络层数据但抽样率更大并且主要监测用户端播放全过程的用户体验。

2.2.2.1 PC端到端真机监测

自从1991年全球第一个网站上线以来，互联网经过25年的蓬勃发展，已经与人类日常生活息息相关。根据Netcraft发布的Web服务器调研数据显示，截止到2015年12月，全球共有9.01亿网站在

线。在“互联网+”的时代背景下，用户体验是最终检验企业是否成功最好的标准。而网站和网页加载速度是决定用户体验的关键指标。访问速度下降不仅带来用户流失，更会影响企业的业务，造成营收的损失。Radware公司公布的调研结果显示，加载时间每提升1s，沃尔玛官网的购买转化率将提升2%。Firefox平均加载时间每减少2.2s，下载次数将提升15.4%，每年将增加预计1000万次下载。如果汽车配件零售商AutoAnything加载时间减少一半，其销售额将提升13%。

通过有偿招募的方式在真实用户的PC、手机上安装监测客户端收集性能数据是目前PC监测中最常见，也是监测数据最完整的监测方式，可以通过服务端下发监测任务到客户端，按一定频率主动监测产品速度和体验，并发送到服务器进行展现和分析。第三方商业公司做得较好的有gomez、keynote、基调网络等公司，最大的特点是可以监测竞争对手。显然客户端主动监测与JS被动监测各有利弊，相辅相成。通过本人在腾讯、百度的实践，这两种监测必须具备，并建立真实用户速度监测体系，为速度优化提供依据。通过客户端监测国内外、各终端、各浏览器下各产品形态，在前端、网络、系统、应用层的性能及针对性地进行优化，目前百度搜索产品、商业产品经过监测、分析、优化后，已经达到业界最快。

PC真机监测价值及目标

- 1 建立PC真实用户性能监测体系，通过分布全国的真实监测客户端，采集浏览器访问网站过程中的各项性能指标，从渲染层的白屏时间、首屏时间等指标，到网络层的DNS解析时间、建立连接时间、SSL时间、重定向时间、首包时间、总下载时间等指标，指标全面丰富，从各个角度衡量网站性能。同时提供运维决策支持，通过对监测数据多个维度的分析，为产品线研发、运维容量规划、容灾备份、迁移扩容提供数据支撑。
- 2 分析网站性能瓶颈，指导性能优化，提供从终端用户侧发起真实Web访问的性能监测数据，通过监测数据对网站及应用进行合理优化，快速迭代，加快开发进度，提高用户访问速度。快速定位故障，精确到具体影响省份、城市、运营商，在最终用户反馈前解决问题。同时支持竞品对比分析，对比竞争对手应用性能，了解自身所处行业位置，通过分析监测数据，缩短与对手差距，保持行业领先地位。
- 3 交互性能事务级请求分析，支持在网页监测任务中进行登录、点击、选择等事务操作的定制化需求，通过事务脚本引擎ScriptEngine与浏览器内容进行深度交互，完成事务操作，并采集操作过程中各种响应时间，反馈真实用户在与网页交互时的真实性能体验，可以具体定位到某一个或多个Ajax API调用响应慢。使用动态图像扫描分析技术(ScreenShot)+视频录制技术(VideoCapture)真实还原网页打开过程中每一帧画面的变化，最大程度再现网页加载过程，获取白屏时间、首屏时间等渲染层核心指标，帮助用户迅速发现渲染层问题。

- 4 监测数据开放API，对外提供数据开放数据API，支持线上监测数据实时报表导出，产品线通过API调用文档，也可方便地获取监测源数据，进行监测报表的自定义设计与开发，对数据具有二次开发定制需求的用户可灵活使用监测数据。

PC真机监测实现原理

- 1 页面监测，监测点客户端启动浏览器打开指定Web页面，动态加载BHO（浏览器辅助对象）插件实现网络层时间指标的采集，该插件实现了WinInet API的接口InternetStatusCallback，通过在网络通信不同阶段记录起止时间戳，从而获取DNS解析、建立连接、SSL握手、首包、剩余包、加载时间等时间指标。
- 2 文件监测，通过扩展Windows的浏览器封装类CHtmlView，实现了IDownloadManager的Download接口，因此当浏览器打开指定页面下载时，进入自定义下载接口自动下载，从而实现文件下载类监测。
- 3 事务流监测，通过C++与webbrowser浏览器中JS交互操作，将事务转换为JS脚本形式，通过webbrowser的scriptEngine实现登录、点击等事务操作效果。
- 4 渲染监测，通过CxImage实现的ScreenCapture截图器，从页面加载开始进行连续截图，与浏览器StartRender，DocumentComplete等事件结合获取首次渲染时间、首屏时间等渲染层和性能指标。

PC真机监测拓扑图如图2-10所示。

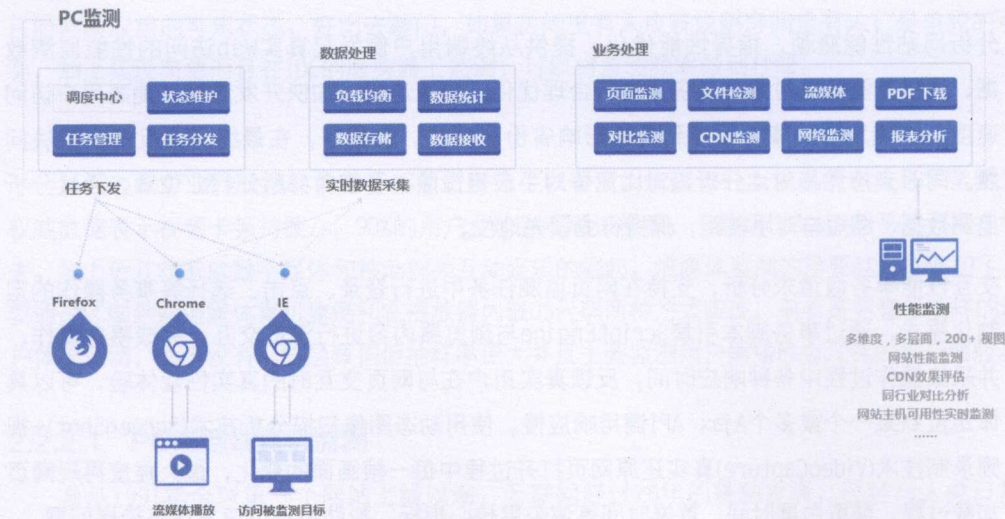


图2-10 PC端到端真机监测拓扑图

PC真机监测架构及说明如图2-11和表2-6所示。

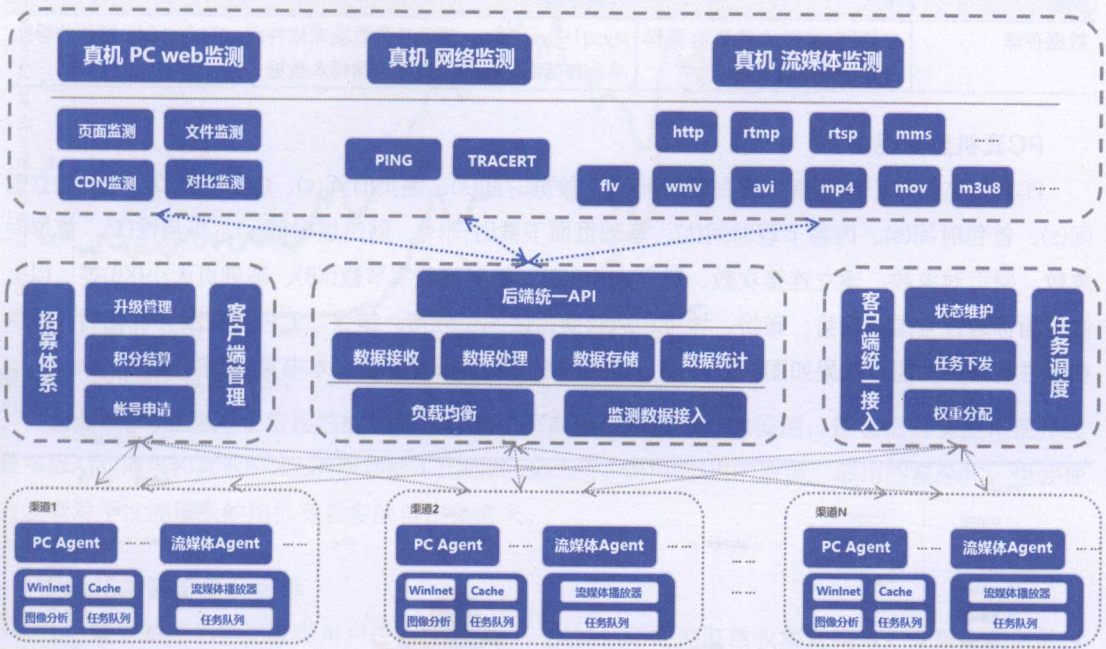


图2-11 PC端到端真机监测架构图

表2-6 PC端到端真机监测模块及功能

模块	作用	实现原理
PC监测Agent	执行Web真机/网络/文件监测任务	使用WinInet/缓存控制/图像处理/任务队列等技术，采集用户端访问指定网页的真实用户质量数据
流媒体监测Agent	执行流媒体真机监测任务	嵌入流媒体播放器，使用hook/网卡抓包捕获等技术，采集DNS解析时间/建立连接时间/首包时间/缓冲时间/缓冲次数等播放性能数据，支持http/rtmp/rtsp/mms等多种网络协议，支持avi/flv/wmv/mp4/mov/m3u8等多种流媒体格式文件播放
招募体系	监测客户端账号申请/积分结算/升级管理	Nginx+yaf管理招募用户账号，按各规则（时间/任务）结算招募用户积分，通过fastdfs平台对所有接入Agent进行在线升级
客户端管理	统一配置管理各渠道接入的Agent	Nginx+php-fpm的cgi接口全局统一分配Agent的接入ID，Redis存储管理Agent的工作/健康状态，对Agent进行统一管理
客户端统一接入	与各监测Agent建立TCP长连接，维护Agent状态	基于开源软件mosquitto二次开发，使用内存索引技术维护Agent在线状态数据
任务调度	向PC/流媒体监测Agent下发监测任务	使用fastcgi向指定索引簇的Agent组通过mosquitto建立的长连接下发任务
数据接收处理	接收PC/流媒体监测Agent回传的监测数据	使用Nginx+php-fpm接收并解析原始数据，得到一次监测样本中的各性能指标，调用IP解析/localdns解析等服务

续表

模块	作用	实现原理
数据存储	存储PC/流媒体真机监测样本数据供前端展现	mysql+fastdfs，使用开源数据库软件mysql+fastdfs搭建的静态平台存储Web/网络/流媒体监测样本数据

PC真机监测视图

PC真机监测核心性能指标有白屏时间(s)、首屏时间(s)、整页时间(s)、DNS时间(s)、连接建立时间(s)、首包时间(s)、内容下载时间(s)、基础页面下载时间(s)、网络层时间(s)、可用性(%)、首屏对象数、网页对象数、建立连接次数、页面大小(KB)、首屏下载字节数(KB)、基础页大小(KB)等。以上性能指标通过全国、趋势、省份、城市、运营商、区间、分布、错误、汇总等视图可视化后，提供在线性能分析功能，效果如图2-12所示，详细的视图和讲解说明会在本书第5章中介绍。

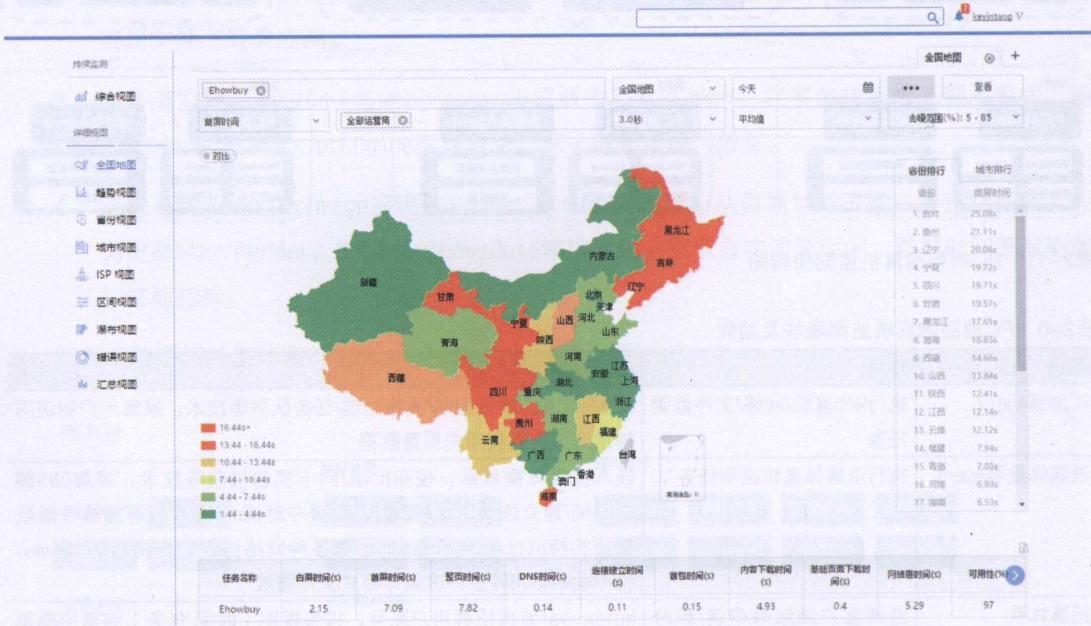


图2-12 PC真机监测产品视图

2.2.2.2 PC JS监测

在2015年的国际范围内，Web开发领域正向构建更复杂和工程化的大型企业级应用的方向发展，技术领域涌现出ReactJS和AngularJS等代表的 MVC 框架。同时受移动化趋势的影响，基于移动后端服务（MBaaS）解决方案和API驱动开发，也使得单页应用（SPA）变得更加流行，以便与移动应用共用同一套后台系统。从国际职位搜索引擎Indeed提供的最新职位趋势分析看，2015年企业对Web应用及相关技能的招聘需求大幅度增长，如图2-13所示。

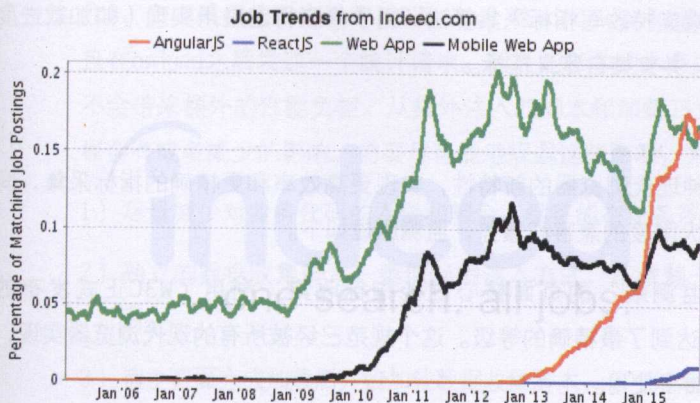


图2-13 Indeed职位趋势统计

企业级Web应用对于应用性能的需求，除了页面复杂加载过程的监测，还包括对于应用使用过程中以Ajax请求和其他HTML5的新功能（比如websocket）调用的API的监测。应用的复杂化，也会使开发者对于性能采集的指标有更多的自定义需求。

PC JS监测价值及目标

- 1 建立基于JS的PC真实用户性能监测体系，支持从不同浏览器收集真实用户使用应用时的性能参数，反映真实用户体验。结合应用系统整个拓扑结构各层次节点的监测数据，交叉参考，提升对性能问题快速发现、精确定位、迭代优化的能力。
- 2 丰富的数据分析功能，有效分析不同地域、浏览器、系统上用户打开页面的体验；并支持任务间的交叉比对，可应用于灰度发布和测试等领域；代码级问题定位，在JS错误、慢页面、资源加载失败等问题跟踪时，提供包括脚本文件、行数、终端环境等丰富信息，帮助精确定位和解决问题。
- 3 监测产品环境下使用的第三方API和资源，通过在浏览器端采集数据，能够监测常见的第三方API和资源的服务质量，如第三方登录、流量分析工具、CDN等，帮助及时发现第三方问题，保证产品可用性。
- 4 轻量级监测端安装方式，充分利用各Web平台的兼容性，用最小的工程代价，快速实现最大范围和类型的应用入口的覆盖，从简单的图文网页，到复杂的单页应用。
- 5 多维度的性能指标，即支持页面全局的DNS解析时间、用户可操作时间、整页时间等指标，也支持页面内服务器端和第三方资源载入监测，还支持页面JavaScript发起的Ajax请求的监测以及代码错误收集。既适用于监测内容为主的页面，也适用于监测交互丰富的单页应用。

- 6 监测端的可扩展性，监测端支持改写指标采集算法，用于修正特定应用实现（如加载进度指示等）的真实指标数值，并支持自定义指标。

PC JS监测实现原理

JavaScript监测脚本利用了多种现代浏览器的新特性，实现更高效率和更精确的指标采集，同时保留了对流行的老版本浏览器最大限度的兼容和覆盖，监测原理如下。

- 1 现代浏览器的特性，“监测端”的页面核心性能指标收集，利用了W3C正式发布的Navigation Timing规范，达到了最精确的等级。这个规范已经被所有的现代浏览器实现，能够在所有智能手机浏览器中使用。
- 2 浏览器的JavaScript API监测，现代Web应用采用了大量新的浏览器API，比如说Ajax请求。“监测端”利用了较新的浏览器中Object.defineProperty接口，实现对大多数浏览器API调用的代理和监控。这样既不影响应用代码的正常运行，同时不需要开发者为监测额外进行适配。
- 3 传统的时间戳收集和回传方法，在使用现代浏览器的新特性同时，“监测端”也保留传统实现中的时间戳收集方式，在“首字节时间”、“白屏时间”等指标上达到了最大兼容性和覆盖面。并且通过对图片元素注册事件监听，安全有效地实现“首屏时间”等指标收集。在数据回传上也根据浏览器环境使用多种回传接口，确保数据在各种条件下完整高速回传。
- 4 配置下发模块搭配使用CDN服务，可以允许“任务管理”模块动态地更新监测端配置，并通过缓存刷新、设置过期时间等内部API，以分钟级速度下发大流量的配置。

PC JS监测的一些细节

以嵌入JS到被测页面中的方式，对网页进行大采样量的真实访问体验监测，并且可提供各种浏览器和操作系统下的用户体验数据。通过在网页中内嵌JS监测代码，当用户访问有JS监测代码的网页时，会收集用户的速度和体验信息，发送到服务器端进行展现和分析。不同于真机性能监控方式，不再需要大规模在各省市部署监控点，不再需要复杂的自动化脚本，不再需要一堆浏览器插件。另外，充分利用JavaScript的特性，采集用户的行为数据，实时了解到用户喜欢什么，不喜欢什么，帮助产品和业务更好地了解用户，改善营销效果，提升用户体验。JS监测的一些细节介绍如下。

- 1 JS监测对页面的影响，采集脚本会以内联的方式注入到页面中，会进行随机采样。命中采样时会额外加载一个JS文件，进行数据收集和回传的操作。对页面的影响在于页面的前后各增加一段script标签，页面中注入约3.7KB（GZIP后约1.9KB）的代码，导致数据传输增加

2KB, 命中采样的用户 (约100万) pv, 在window.onload之后需要额外加载一个JS文件, 并且在onload之后发送一个统计请求。因性能脚本的本身逻辑非常简单, 执行在毫秒级别, 不会带来额外的性能负担。从额外注入JS脚本和加载JS文件的角度, 对页面加载和渲染过程会有或多或少的影响, 需要尽可能地规避这些影响, 常用的方法如下。

- 1) 尽量减少对业务代码的入侵和耦合, 尽可能在业务逻辑之外。
 - 2) 独立的数据收集脚本, 类似GA的注入方式, 不依赖于前后端框架, 方便移植到其他项目。
 - 3) 宿主页面完成加载后开始加载数据收集脚本, 尽可能地合并性能数据的发送, 节省带宽。
- 2 JS监测对浏览器的兼容情况, Navigation Timing是用于在页面中精确测量性能的API, 获得了比较多的现代浏览器支持, 同时也被多款性能检测工具应用于实现页面采集数据, 比如Google Analytics (分析), 兼容情况如图2-14所示。

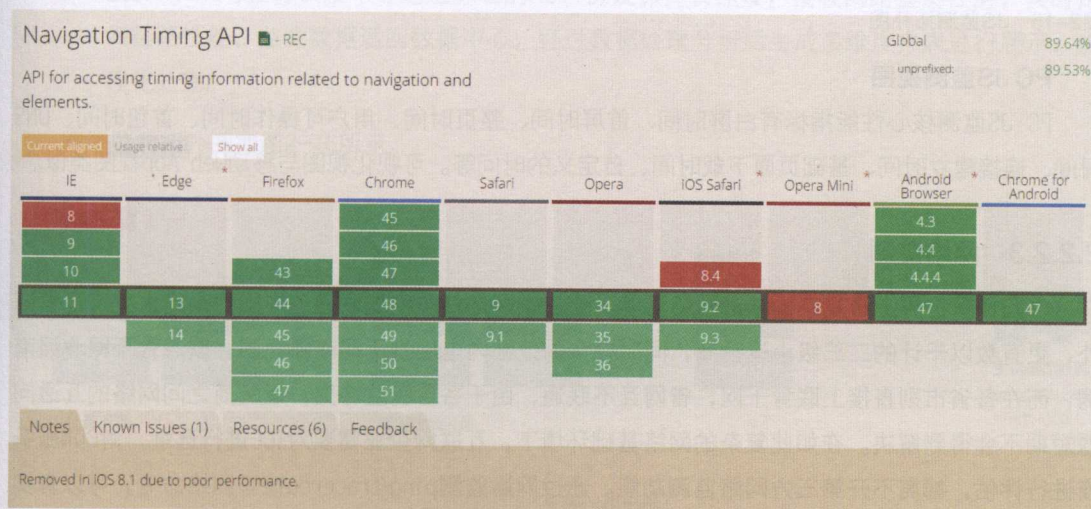


图2-14 Navigation Timing兼容统计

PC JS监测拓扑如图2-15所示。

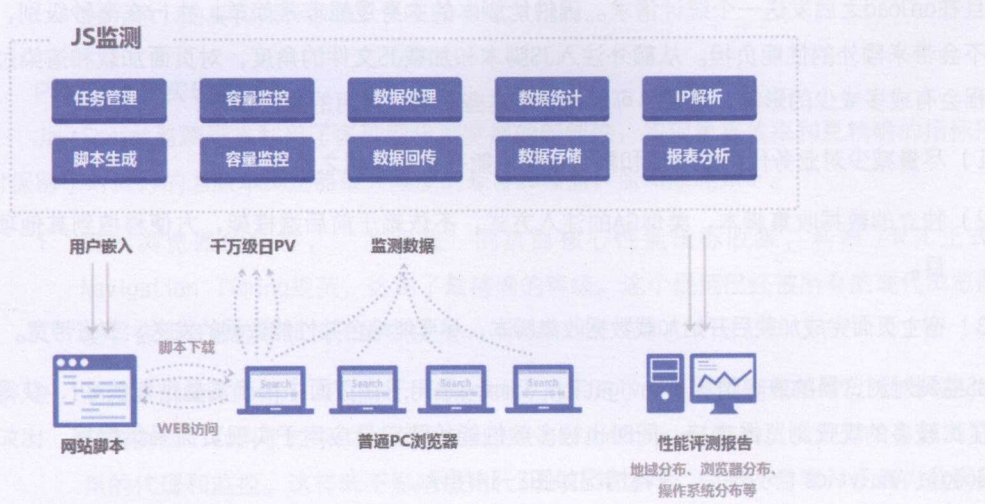


图2-15 JS监测拓扑图

PC JS监测视图

PC JS监测核心性能指标有白屏时间、首屏时间、整页时间、用户可操作时间、首包时间、DNS时间、连接建立时间、基础页面下载时间、自定义的时间等。可视化视图与移动Web App视图类似。

2.2.2.3 网络监测

中国拥有世界上最复杂的基础网络环境，除去中国电信、中国联通、中国移动三大主流运营商外，更有数以千计的二级小运营商，同时在全国范围内电信骨干网南强北弱，联通骨干网北强南弱，而在各省市则直接上联骨干网，省网互不联通，由于各自利益冲突，运营商之间网络的互通问题短期不会得到解决。在如此复杂的网络基础环境下，互联网企业需要对IDC进行选址，对CDN服务商进行评估，都离不开第三方网络监测功能。通过网络监测ping/traceroute等网络命令，可以真实感知各服务厂商提供服务的网络延时，同时为企业选择服务提供直接的数据支持。

网络监测价值及目标

- 1 提供全面的网络性能监控，实时调用全国真实用户本地监测Agent对目标域名或主机IP进行链路探测，返回ping/traceroute真实数据。Agent调用PC本地ping/traceroute等网络探测命令，可在执行Web监测任务的同时探测网络链路性能，也可单独监测链路性能。
- 2 为客户IDC/CDN服务选型提供对比数据，了解各供应商的服务性能优劣，确保投入性价比最大化。

- 3 提供长期稳定监控数据，当网络发生故障时能及时排障，定位故障地点与原因。
- 4 可以从性能指标如延时、丢包率、可用性等角度分析域名或IP、API的性能，通过趋势视图、散点视图等可视化图形报表提供强大的数据分析功能。
- 5 可以从区域角度查看域名或IP、API的整体性能，通过全国地图、省份视图、城市视图、运营商视图等图形报表提供数据对比与分析。

网络监测实现原理

- 1 监测客户端调用Windows API中的CreateProcess函数，直接启动Windows系统自带的cmd命令，在命令行中执行相应的ping/traceroute等命令获取网络监测结果。
- 2 客户端与服务端之间采用MQTT标准协议连接，维持长连接心跳机制，为防止网络抖动等因素，实现断线自动重连功能，保持服务端对客户端的任务可达性。
- 3 监测Agent与任务调度中心通过标准的MQTT协议保持长连接，接收网络监测任务，实时探测链路性能，监测数据返回数据中心，经过数据处理分析后生成多维度报表进行展示，流程化处理。

网络监测拓扑图如图2-16所示。



图2-16 网络监测拓扑图

网络监测视图

网络监测核心性能指标有Ping延时(ms)、Ping丢包率(%)、Traceroute延时(ms)、Traceroute跳数、可用性(%)等。以上性能指标通过全国、趋势、省份、城市、运营商、区间、散点等视图可视化后，提

供在线性能分析功能，效果如图2-17所示，详细的视图和讲解说明会在本书第5章中介绍。

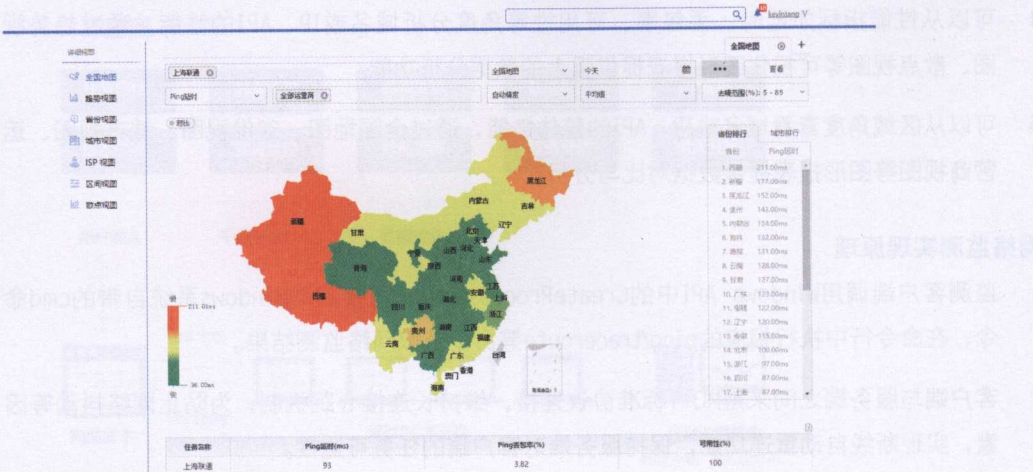


图2-17 网络监测产品视图

2.2.2.4 可用性监测

CNNIC（中国互联网络信息中心）发布了《第35次中国互联网络发展状况统计报告》，截至2014年12月，中国域名总数增至2060万个，年增长11.7%，根据另一机构的预测，2017—2018年，中国域名将达6600万。中国互联网协会、国家互联网应急中心在北京发布的《中国互联网站发展状况及其安全报告（2015）》显示，2014年全年新开通的中国网站数量约95.2万个，网站总数年增14.1万个。另外根据NetCraft统计，截至2014年9月，全球的网站数达到将近10亿，如图2-18所示。腾讯公布的数据，腾讯内部域名达到10万级别，中国互联网的第二个十年将产生更多网站、域名及应用服务，这些海量的基础服务需要更加轻量级的可用性监测去快速监测并第一时间掌握异常情况。

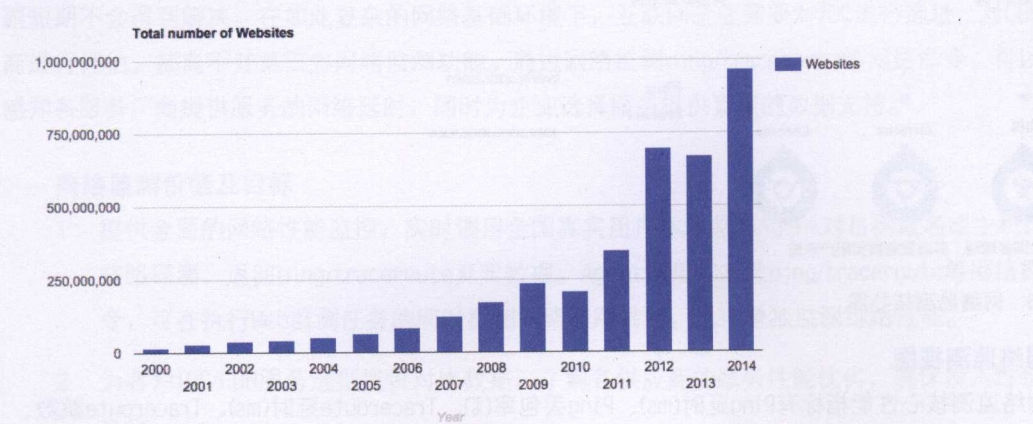


图2-18 NetCraft全球域名统计

可用性监测价值及目标

- 1 建立全面的可用性性能监测体系，几步操作实现监测并覆盖绝大多数产品线，支持秒级HTTP(s)/PING可用性监控。采用标准物联网协议MQTT接入探测客户端，并通过升级支持按照地域、运营商下发探测任务，方便功能扩展。
- 2 使用多个IDC节点监测资源，同时支持电信、联通等主流运营商，不放过任何运营商的网络问题。监测频率最快30s，帮助产品线迅速感知可用性问题的出现以及恢复。
- 3 采用后端统一订阅发布服务，任务下发模块可方便接入，不同渠道探测客户端均可订阅监控任务，同时任务下发也支持按照不同渠道发布任务。采集数据使用MongoDB集群存储，容灾性高，数据支持scheme-free，方便进行结构升级。
- 4 加入监控同时自动添加告警策略，支持告警频控和恢复信息，不需要进行复杂配置。可自定义配置告警组，通过短信或邮件及时通知到相关人员。任务添加以及告警配置步骤简单，并且告警策略灵活，即使是非技术人员也可以轻松操作。

可用性监测实现原理

- 1 Agent订阅，在全国IDC、CDN节点上部署可用性监控Agent，Agent通过实现MQTT订阅/发布协议的mosquitto lib在服务端进行订阅，接收服务端的任务，订阅端会按照Agent IP，对应按照运营商和地区进行索引。
- 2 任务下发，用户创建可用性监控任务之后，服务端会保存监测的域名/URL、监测运营商、监测类型、监测频率，同时将与告警策略相关数据注册至mmTrix消息中心。服务端定期读取任务列表，以及当前订阅的Agent，通过MQTT协议向Agent按照任务发送多个样本采集任务消息。
- 3 可用性探测，Agent接收到任务消息之后，使用icmp协议和libcurl库对于域名或者URL来采集各项网络指标。
- 4 数据上报，Agent与接收端保持长链接，与服务端保持通信，将采集的网络指标数据以及错误信息发送至服务端，服务端将数据存储至MongoDB。
- 5 异常判断，服务端接收到数据之后，根据告警策略，判断样本状态以及结果一次任务下发的多个样本，判断本次任务下发的状态。

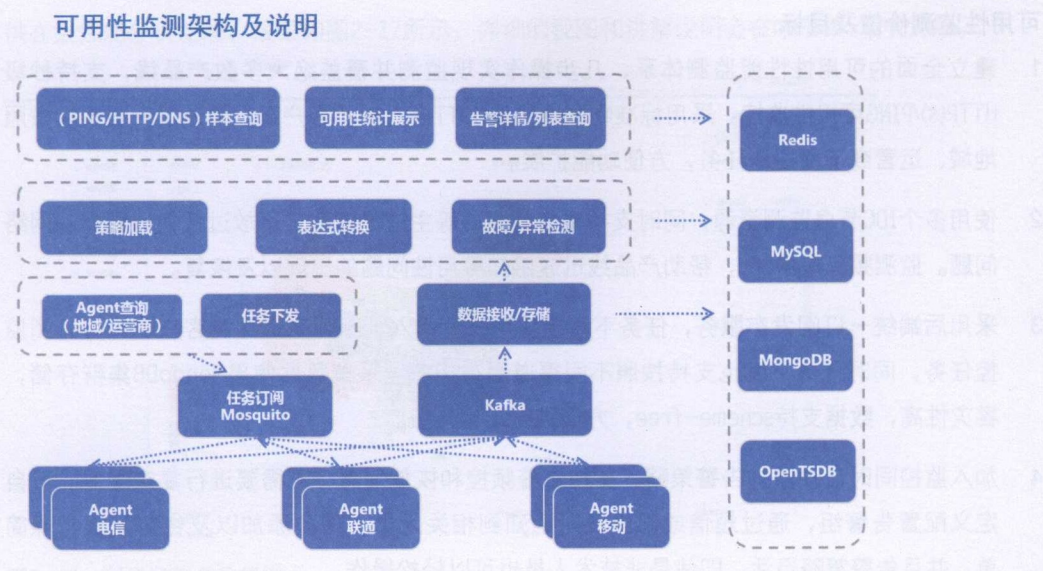


图2-19 可用性监测架构图

表2-7 可用性监测模块及功能

模块	作用	实现原理
可用性监控Agent	订阅topic，获取不同任务的可用性检测请求，并将检测结果返回至服务器	在CDN节点上部署了可用性监控Agent，Agent根据所在节点的地区以及运营商，通过mosquito订阅发布库，订阅不同的任务topic。获取任务后，根据任务类型和参数，执行ping/http/dns等操作，获取检测对象的可用性数据，之后发送至服务器端
任务下发模块	根据用户创建的可用性监控任务，定期将任务请求下发至配置的地区和运营商	定期加载MySQL中的可用性监控任务，根据任务配置的地区和运营商，通过Agent查询模块找到对应的Agent topic，之后周期性向该topic发布消息
数据接收/存储模块	接收可用性监控Agent发送来的数据，将数据进行持久化并转发至异常检测模块	利用kafka接收Agent传来的样本数据，将原始数据存储至MongoDB（后续计划迁移至OpenTSDB）
异常检测模块	根据用户录入的策略，检测Agent数据是否异常	定期从MySQL数据库中加载用户创建的异常检查策略，将策略转化为一系列表达式。长链接接收来自上游模块的可用性监控数据，并结合一次任务下发的其他样本数据，将数据代入表达式计算，判断该次检测是否有异常。如果有异常，将相关数据发送至消息服务中心，告知用户
数据查询模块	按前端要求检索并统计监测数据	根据前端的时间和查询条件，从MySQL和MongoDB中获得监测原始数据，并按要求进行统计和格式化

可用性监测视图

可用性监测核心性能指标有响应时间(ms)、可用率(%)、异常发生时间、异常恢复时间、异常持

续时间等。视图如图2-20所示。

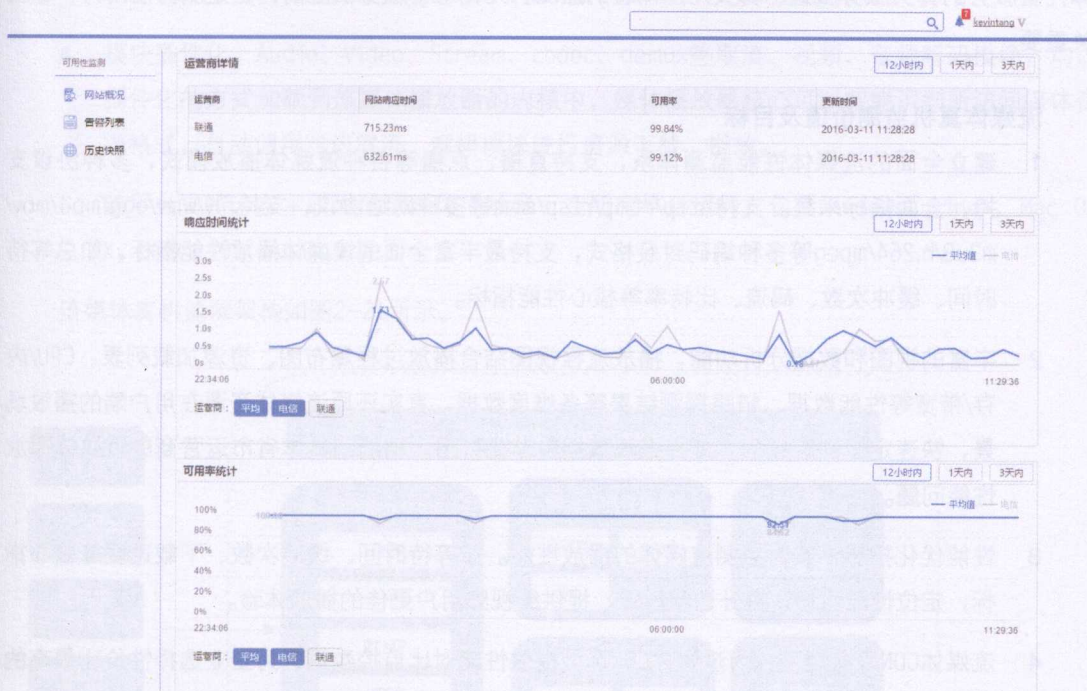


图2-20 可用性监测产品视图

2.2.2.5 流媒体真机监测

美国麻省理工大学教授拉梅什·西塔拉曼(Ramesh Sitaraman)研究了670万独立用户的2300万次视频观看活动,结果显示对很大一部分用户来说,如果视频无法在2s内完成加载,那么这些用户将放弃观看该视频。随着互联网网速的变快,用户的耐心程度正不断下降。视频加载时间每长1s,就会有约6%的人放弃观看。如果视频加载时间达到5s,那么将有约20%的用户放弃观看。随着传统互联网广告的滑坡,视频广告正成为媒体公司眼中的亮点。视频广告能带来更长时间的互动,并有着较高的广告价格。因此提高流媒体视频播放的流畅度,为用户提供最佳的视频播放体验,是企业赢得用户、提高营收的关键。

随着CDN行业产品成本变低,以及乐视云、又拍云、七牛云等提供流媒体托管的新型云服务的出现,中小企业和个人开发者的定制化需求有了更多选择。与此同时,2015年内视频应用类型的创业企业也增长很快,包括在线教育、视频直播、短视频等领域。截止2016年1月,从互联网产品库itjuzi.com的搜索中,围绕“直播”应用的创业企业就达192家。通过CDN市场结构可以看出,视频应用是互联网企业在性能优化方面一个主要需求点。根据中国信息通信研究院发布的《内容分发网

络（CDN）白皮书（2015）》，从CDN业务日均流量看，视频业务流量最多，占比37.5%。对于流媒体托管服务的真实服务质量、真实用户体验的监测，无论对于服务供应商，还是服务使用方，都至关重要。

流媒体真机监测价值及目标

- 1 建立全面的流媒体性能监测体系，支持直播、点播等各种流媒体播放模式，多种协议支持，全面指标采集。支持http/rtmp/rtsp/mms等多种网络协议，支持flv/wmv/ogg/mp4/mov/m3u8/h.264/mpeg等多种编码封装格式，支持最丰富全面的流媒体播放性能指标，如总等待时间、缓冲次数、码流、比特率等核心性能指标。
- 2 丰富的视图和数据分析功能，播放流程视图结合播放过程瀑布图、资源加载列表、CPU/内存/带宽等性能数据、链路探测结果等各维度数据，真实还原流媒体资源在用户端的播放场景，快速定位性能瓶颈。支持报表数据自动化导出，精确到具体省市运营商的流媒体播放性能问题。
- 3 性能优化指导，实时监测流媒体的播放性能。如等待时间、缓冲次数、下载速度等核心指标，定位性能瓶颈，提升访问性能，提供给视频用户更佳的播放体验。
- 4 流媒体CDN服务选型，为流媒体CDN选型提供性能对比监控数据，为企业选择性价比最高的流媒体CDN加速服务。
- 5 流媒体竞品对比分析，全面了解自身和竞争对手的流媒体播放性能优劣，明白自身流媒体业务性能在行业中的表现，缩短与对手之间的差距，做到行业最好。
- 6 数据API支持，提供监测数据报表自动导出功能，同时提供监测数据对外开放API，参考API文档即可方便获取监测数据，满足有其他定制需求的自由灵活地使用监测数据。

流媒体真机监测实现原理

- 1 流媒体监测端从任务调度中心获取监测任务并播放，通过对系统层网络通信API调用的监听，深入系统层监测到DNS解析、TCP建联、首包等网络层时间消耗，详细记录从DNS解析到数据传输各阶段性能指标。
- 2 通过播放器内部分流器将音频、视频信息从媒体流中解复用。自动调度取流、编解码模块播放各种流媒体视频，支持HTTP/FTP/RTMP/RTSP/HLS等各种媒体资源的访问，自动调用相应的取流插件从服务端下载媒体文件流，然后使用编解码插件进行播放输出，同时进行各项播放性能指标的采集和处理。

- 3 对媒体流进行软件解码或硬件解码从而输出音频和视频信息，在各解码函数中植入记录代码标记，从而获得流媒体播放过程中的缓冲时间、缓冲次数等关键指标。
- 4 模块插件化，Audio、Video、Stream、codec、demux等取流、视频、音频解码模块，均以插件化的方式加载到流媒体播放器的内核中，媒体播放器核心可以智能识别所访问媒体资源格式，自动调用对应取流、解码模块进行资源下载、播放。
- 5 跨平台支持，自主研发的流媒体播放器可在所有平台运行，包括Windows，Linux，Mac OS X，UNIX，iOS，Android等。

流媒体真机监测架构如图2-21所示。

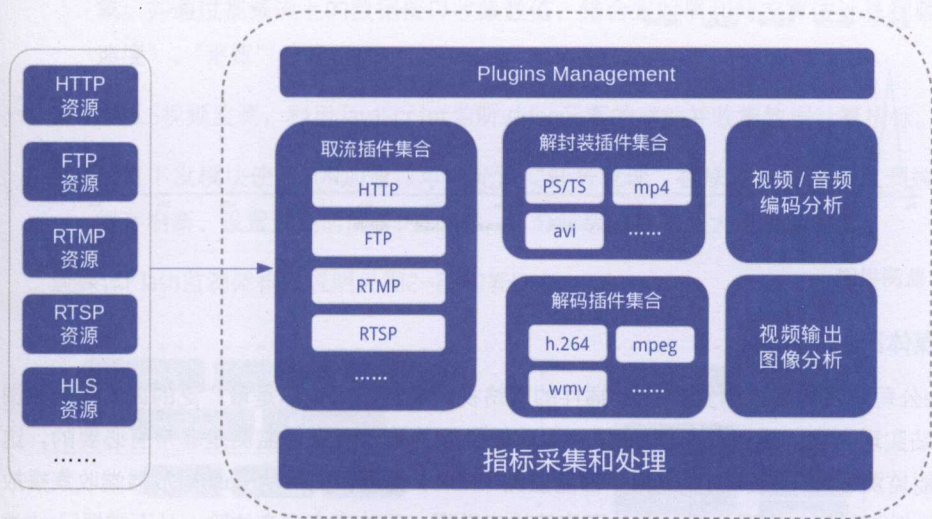


图2-21 流媒体监测架构图

流媒体真机监测视图

流媒体真机监测核心性能指标有总等待时间(s)、视频准备时间(s)、再缓冲时间(s)、DNS解析时间(s)、TCP建联时间(s)、发送请求时间(s)、等待响应时间(s)、首次缓冲时间(s)、协议建联时间(s)、首屏时间(s)、比特率(Kb/s)、码率(Kb/s)、监测时长(s)、播放时长(s)、缓冲前准备时间(s)、首次播放时长(s)、再缓冲次数、可用性(%)等。以上性能指标通过全国、趋势、省份、城市、运营商、区间、错误、播放流程等视图可视化后，提供在线性能分析功能，效果如图2-22所示，详细的视图和讲解说明会在本书第5章中介绍。



图2-22 流媒体监测视图

2.2.2.6 流媒体Flash监测

目前Adobe公司开发的围绕浏览器Flash插件的网络视频播放技术依然是最广泛的面向PC端和浏览器的视频网站实现方案。通过基于Flash插件实现轻量级的被动流媒体监测是非常有必要的，这类监测也是性能监测领域中强调的真实用户性能监测（RUM）类型，支持从Flash播放器端收集播放量、首屏时间、播放时间、地理位置、播放深度等性能数据，反映真实用户体验。从而掌握Flash类流媒体资源在不同网络条件下的访问情况，实时了解用户观看体验，从而帮助企业发现播放过程中的问题，为改善播放效果提供数据依据。

流媒体Flash监测价值及目标

- 1 建立基于Flash播放器的流媒体性能监测体系，基于JavaScript和Flash，支持从不同浏览器收集真实用户观看视频时的性能参数，从播放器端收集记录播放量、首屏时间、播放时间、地理位置、播放深度等性能数据，反映真实视频用户体验。
- 2 多种监测端安装方式，支持不同视频播放方案，以轻量级代码的方式，简单快速适配多种源播放器，能够从不同播放方案下采集相同数据，如同时支持PC版的Flash播放器和移动版的HTML5播放器。并支持从视频网站HTML嵌码转发方式，到开源播放器和自建播放器。
- 3 丰富的数据分析功能，有效分析不同地域用户打开页面的体验，并支持任务间的交叉比

对，可应用于灰度发布和测试等领域。

- 4 监测和评估第三方流媒体CDN服务质量，通过所有客户实际播放行为采集的性能数据，评估所使用的第三方流媒体CDN服务的真实质量，有效验证投资收益。

流媒体Flash监测实现原理

- 1 Flash封装，类似Flash广告、弹幕播放器等技术实现原理，利用FlashVars从HTML标签中传入原播放器地址和参数。
- 2 Flash视频监测，利用Adobe官方推荐监测接口NetMonitor，注册监听Flash播放器中的NetMonitorEvent.NET_STREAM_CREATE事件，获取全局每一次创建的视频流NetStream对象。并通过视频流上的数据接口收集数值，结合定时器和特定算法计算视频播放的“下载速度”、“带宽”等指标。
- 3 HTML5视频监测，利用JavaScript监听video元素的事件并收集数据计算指标。
- 4 配置下发模块使用CDN加速，可以允许“任务管理”模块动态地更新监测端配置，并通过缓存刷新、设置过期时间等内部API，以分钟级速度下发大流量的配置。

流媒体Flash监测架构及说明见图2-23和表2-8。

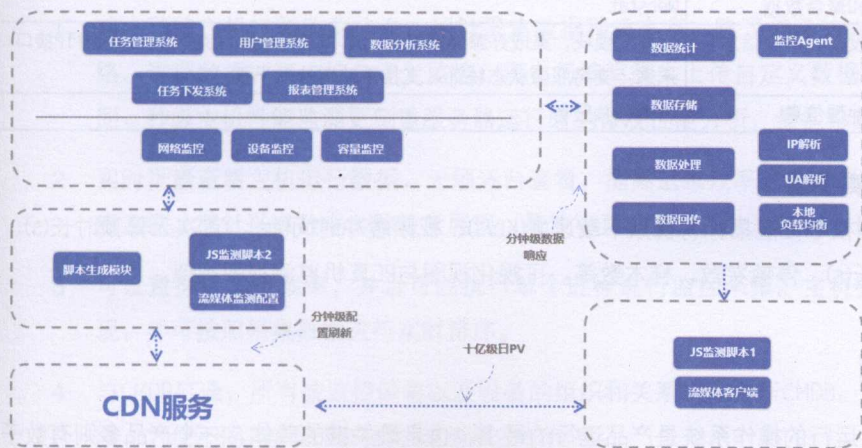


图2-23 流媒体Flash监测架构图

表2-8 流媒体Flash监测模块及介绍

模块	作用	实现原理
任务下发系统	调用脚本生成模块	前端通过HTTP API调用和传递参数
数据分析系统	查看监测任务指标数据	前端通过HTTP API调用webstat接口获得数据，在页面展示图形图表

续表

模块	作用	实现原理
脚本生成模块	生成、部署、更新脚本和配置文件	Nginx实现简单HTTP API，根据模板生成配置，Etcid同步多个节点，对外通过CDN服务下发。离线脚本定时根据采样量更新配置，也可手动干预。 未来：支持HTTPS，自动化精细化CDN的缓存管理，配置模板版本化，操作API化
JS监测脚本1	任务部署、配置读取、部分指标监测	JS脚本、监听浏览器特定事件 未来：代码模块化，框架化，功能可配置，集成业界标准的监测项目
JS监测脚本2	部分指标监测、采样率计算、数据回传	JS脚本、通过图片标签跨域回传数据、使用Navigation Timing API 未来：探测设备环境、持续监测、功能插件化、支持扩展
流媒体客户端	任务部署、配置读取、指标监测、数据回传	AS3开发，独立SWF文件来导入用户播放器，使用AS3类接口读取视频流数据 未来：支持常见开源播放器，支持HTML5视频，支持HTTPS，支持Flash嵌入代码部署，支持参数化编译SWF文件
流媒体监测配置	配置信息，如采样率、算法等	带格式文本文件，由客户端通过HTTP读入
数据回传	接收记录回传数据	多台HTTP服务器负载均衡，使用accesslog记录数据 未来：使用kafka等消息队列替代
数据处理	读取和解析回传数据	离线定时任务，调用IP解析模块、UA解析模块及预定义算法，计算指标数据，入数据库 未来：使用分布式计算集群，如MapReduce、Storm替代。支持持续监测和多请求样本类型
数据统计	读取和聚合数据	Webstat
UA解析	确定设备和厂商信息	Go开发模块，集成开源UA解析规则库和UA样本库，支持Thrift和HTTP接口 未来：支持服务状态统计，支持手动增补及干预
IP解析	确定地理信息	后端公共服务

流媒体Flash监测视图

流媒体Flash监测核心性能指标有视频下载速度(Kb/s)、视频缓冲时间(ms)、首次无停顿时长(s)、带宽(Kb/s)、播放时长(s)、停顿次数、样本数等。可视化视图与PC真机监测视图类似。

2.2.3 系统监测

服务器及在之上运行的操作系统是产品运行的最小，也是最关键的载体，一个产品多则有数千台服务器分不同模块，分布在不同IDC构成，这些服务器上的运行状态和性能也直接关联到产品及使用产品的用户的体验。服务器状态和性能主要通过监测操作系统运行状态和基于操作系统的应用、网络状态实现。中小企业都使用开源的系统监测工具Nagios、Zabbix、Monit等实现，这些工具已经开源较长时间，而且不断得到完善，大企业都是自主研发来应对规模化和多样性的挑战。系统监测不像多数人想象的那样只是服务器状态的监控，更多的是通过系统监测承上启下掌握与系统关

联的所有系统、应用、网络、硬件及之间的状态，进一步可以实现更深度的资源、调度、容量管理和自动化运维相关的工作，例如自动部署、扩缩容等。

2.2.3.1 主机监测

服务器出货量持续增长，中国市场尤为突出。互联网催生了中国市场规模化的数据中心的立项和建设，大规模乃至超大规模数据中心的建设越来越热，全面带动服务器出货量的提升。根据IDC统计，2015年第二季度，全球服务器出货量约为229万，较同期增长了3.2%。中国x86服务器市场同比增长8.67%，环比更是提升12.03%。根据whd.global的预计，2017~2018年，中国国内的主机量将达到2640万。

公有云方面，最具代表性的服务商亚马逊于2015年4月首次公布AWS的业绩：2014年收入51.6亿美元，2015年1季度AWS收入15.7亿美元，年增速超40%。2015年全年的营收超过70亿美元，并且根据营收预计，2015年AWS的虚拟机在大约800万的级别。另外，根据美国BLS（劳动统计局）的数据，计算机与IT相关的职位预计从2014到2024年将有12%的增长，超过其他所有行业的平均值，从业者将从390万增长至440万。加上中国“互联网+”国策，国内主机数量将会不断放大，主机性能的监测及优化也变得越发重要。

主机监测价值及目标

- 1 建立秒级主机性能监测体系，探针采集数据种类丰富，除了普通的内存、CPU、磁盘、网络、进程数据，还支持自定义监控，可自行编写脚本上传自定义数据。与传统系统监控不同，秒级主机性能监测更侧重服务器运行时的微观性能分析，最快诊断并确认性能问题。
- 2 实时批量查看主机运行数据，无须逐台查看，提高运维效率。支持自定义集群、模块、分组监测。探针采集数据多种粒度展现，最快可至秒级，不放过任何一个细小的数据异常。
- 3 可设置探针上传频率，并且可以执行单个进程进行监控采集。主机列表综合数据实时展现，并可按照采集数据进行实时排序。
- 4 与CMDB打通，所有被监控设备以及设备的组织和关系均来自于CMDB。可以设置用户查看权限，方便不同用户及用户组共享查看主机信息。
- 5 自定义告警策略，秒级发现主机或者业务异常，并提供完整告警发生/恢复时的信息。基于表达式的告警策略，可多维度灵活帮助用户判断异常和故障。
- 6 高兼容性，探针Agent采用Go语言静态编译，不依赖系统相关库，支持市面上几乎所有Linux发行版本，也方便迁移至Windows平台。

主机监测实现原理

- 1 数据采集，主机监控Agent采集两部分数据，一种是基本数据，基于Linux虚拟文件系统/proc，采集CPU、内存、IO、网络、进程等操作系统指标。另外一种是自己定义数据，Agent开启HTTP端口，用户可以在服务器自行运行采集数据的脚本程序，之后将数据发送至Agent；Agent在下一个数据上报周期，将自定义数据上报至服务端；服务端再根据自定义数据白名单，确定是否将该数据进行存储。
- 2 数据上报，Agent与接收端保持长链接，与服务端保持通信。一方面，定期将采集数据上报至服务端；另一方面，定期向服务端注册主机信息，并获取最新配置，例如注册、上报间隔时间等。另外，接收端采用HAProxy进行TCP协议层级的负载均衡。
- 3 数据异常判断，服务端定期更新用户配置的告警策略，接收到Agent上报的数据，对与该数据相关的策略表达式分别进行判断，如果判断数据异常，则将信息发送至mmTriX消息中心，由消息中心负责拼装模板、发送消息至对应设备，以及进行频率控制。
- 4 数据离线处理，每隔一段时间，服务端将原始上报数据，按照不同时间粒度进行离线处理，以满足不同时间粒度的查询。
- 5 数据展现，数据传输方式上，服务端提供HTTP和Thrift两种方式进行数据展现，HTTP接口提供历史数据，Thrift提供实时数据。展现形式上，除了展现固定的基本数据以外，还支持自定义数据，可以在Dashboard上配置自定义数据以及基本数据的图表。

主机监测拓扑图如图2-24所示。

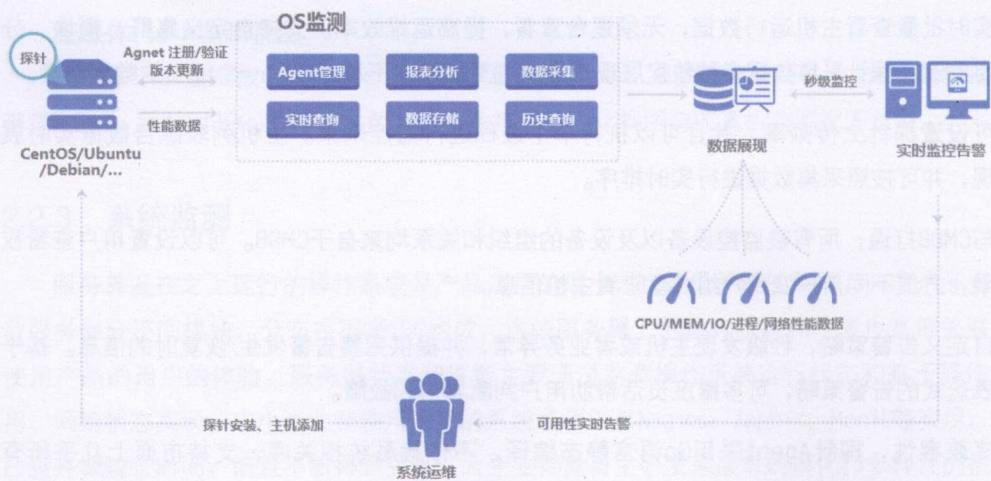


图2-24 主机监测拓扑图

主机监测架构及说明如图2-25和表2-9所示。

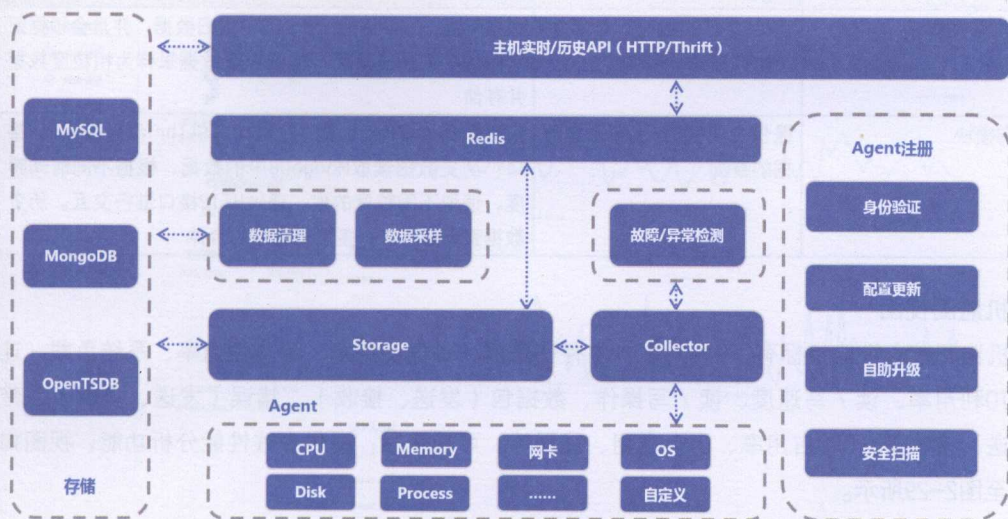


图2-25 主机监测架构图

表2-9 主机监测模块及介绍

模块	作用	实现原理
数据采集Agent	采集服务器的各项硬件指标和性能指标，周期性上传至服务器。另外，还进行身份验证、配置更新、Agent版本更新	通过几种方式进行数据采集：1）利用Linux内核虚拟文件系统proc，从不同虚拟文件中获取硬件及性能指标；2）执行内置shell命令，获取性能统计数据；3）用户在配置中自定义脚本，通过脚本获取采集数据
Agent注册模块	完成新Agent的数据注册功能，检测用户身份和添加主机上限	Agent定期会向注册模块发送心跳数据，注册模块会根据Agent上次的token以及主机名称，验证主机归属。如果是新主机，会在MySQL和MongoDB中进行主机信息的注册
数据接收（Collector）模块	接收来自Agent的上报数据，缓冲并发送至下游的数据存储和数据检测模块	通过维持和Agent进行的长链接，接收来自Agent的msgpack编码的json数据，检测数据格式、用户身份，之后转发至数据存储和数据检测模块
数据存储（Storage）模块	完成采集数据的持久化	通过长链接接收上游模块的Agent采集数据，一方面存储MongoDB用于历史数据查询，另一方面存入Redis用于实时数据查询
异常检测模块	根据用户录入的策略，检测Agent数据是否异常	定期从MySQL数据库中加载用户创建的异常检查策略，将策略转化为一系列表达式。长链接接收来自上游模块的Agent采集数据，并结合该主机的历史数据，将数据代入表达式计算，判断主机数据是否有异常。如果有异常，将相关数据发送至消息中心服务，告知用户

续表

模块	作用	实现原理
数据清理/采样模块	清除过旧的数据，以及对历史数据进行采样	根据配置，定期清除MongoDB中的旧数据；并且会根据配置中的不同采样策略，将细粒度数据采样为粗粒度数据并存储
数据查询模块	提供主机实时/历史采集数据的查询	实时数据读取Redis中的数据，提供Thrift封装的TCP接口；历史数据读取MongoDB中的数据，根据不同时间跨度，读取不同粒度的库，通过HTTP接口进行交互。历史数据查询还可进行多主机数据的合并

主机监测视图

主机监测核心性能指标有CPU利用率、内存使用率、磁盘使用率、带宽使用率、系统负载、连接数、IO利用率、读 / 写速度、读 / 写操作、数据包（发送、接收）、错误（发送、接收）、带宽（发送、接收）、CPU占用率、内存占用、线程等。可视化后，提供在线性能分析功能，视图如图2-26至图2-29所示。



图2-26 主机监测概览视图

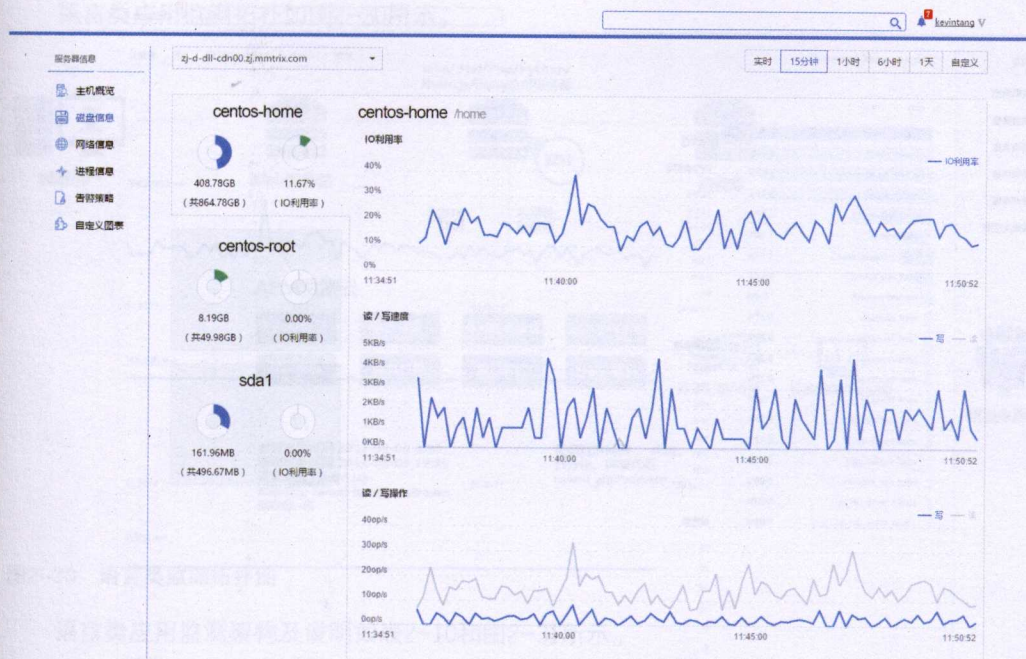


图2-27 主机监测磁盘视图



图2-28 主机监测网络视图

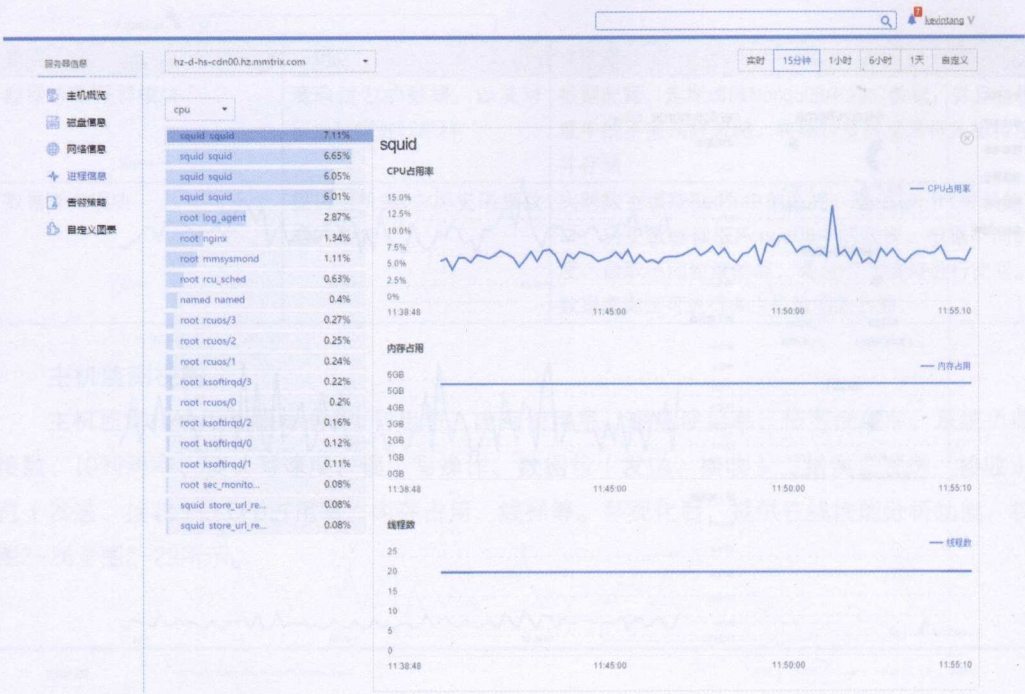


图2-29 主机监测进程视图

2.2.4 应用监测

这里的应用监测主要局限于服务器环境，侧重产品运行的生产环境、功能模块、应用组件等，通过在服务端安装对应的Agent实现在服务器上运行的应用性能监测，进而实现对应用代码、数据库、关联外部服务等性能监控，及时发现应用性能问题并定位性能瓶颈，提供性能问题诊断、追踪及优化依据。与系统监测因环境单一经过多年沉淀百花齐放不同，应用监测存在于商业竞争激烈的大背景下。产品创新使产品架构日渐庞大，应用系统的需求也在快速迭代中不断复杂，应用中的性能问题越来越突出，而这些问题却很难反映在用户层、系统层性能数据上，需要深入到服务器运行的应用环境监测。应用监测主要分为语言和平台两类。

2.2.4.1 语言类监测

语言类是指监测对象为各主流开发语言，并将上下游关联的所有应用性能可视化并进行深度分析，例如调用第三方API、数据库、中间件等，语言类监测主要有Java、Python、PHP、Node.js、Ruby、.NET等。

语言类应用监测拓扑如图2-30所示。

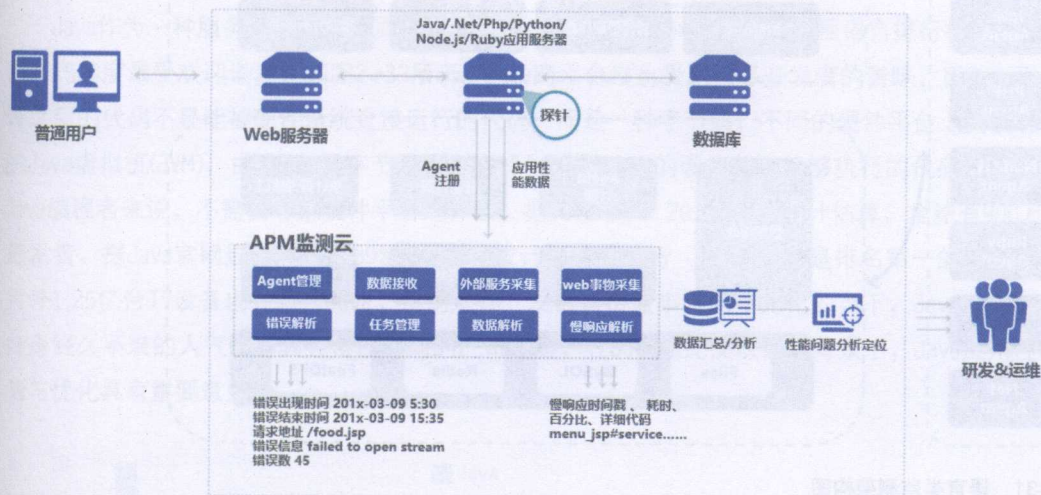


图2-30 语言类监测拓扑图

语言类应用监测架构及说明如表2-10和图2-31所示。

表2-10 语言类监测模块及功能

模块	作用	实现原理
数据接收	接收Agent发送过来的数据，作为Agent和后端的代理	接收Agent数据，保存为本地文件，并通知Agent注册模块或数据解析模块
Agent注册	Agent首次连接需要注册，验证合法性，下发配置信息	获取Agent相关的用户信息、任务信息、本地配置、监控点环境等，验证合法性，并更新配置
数据解析	解析Agent采集的指标数据	解析指标数据，存入数据库，并通知后续的实时分析模块
实时分析	实时分析所采集到的数据，并告警	对比分析实时收集到的信息和近期运行数据，更新状态，发送告警
关联分析	综合分析各项数据	分析各项慢事务数据及JS监测/真机监测/数据库监测的数据，得到其中的关联信息及问题关键点
任务管理	管理用户对任务的增删改及配置	管理任务的增删改及配置，记录于数据库，以便其他模块使用
统计查询接口	根据前端需要，返回统计后的数据	根据前端查询的要求，从数据库中读取数据，进行统计后，返回给前端
任务管理界面	服务器检测任务的注册、分类、查看等操作	调用后端API
数据视图	将Agent采集到的数据可视化展现给用户，分别有总览、Web事务、拓扑、数据库、外部服务、错误等视图	调用后端API，通过echarts将数据可视化
告警管理界面	管理告警策略及通知策略	调用后端API
报表分析界面	通过提供Web事务报表和数据库报表，供用户自行下载分析	调用后端API，生成Excel文件



图2-31 语言类监测架构图

语言类监测视图

语言类监测核心性能指标有Web事务响应时间、外部服务响应时间、数据库事务响应时间、吞吐量(cpm)、错误率(%)等。以上性能指标通过总监、Web事务、拓扑、数据库、外部服务、后台服务、错误、告警、报表等视图可视化后，提供在线性能分析功能，效果如图2-32所示，详细的视图和讲解说明会在本书第5章介绍。

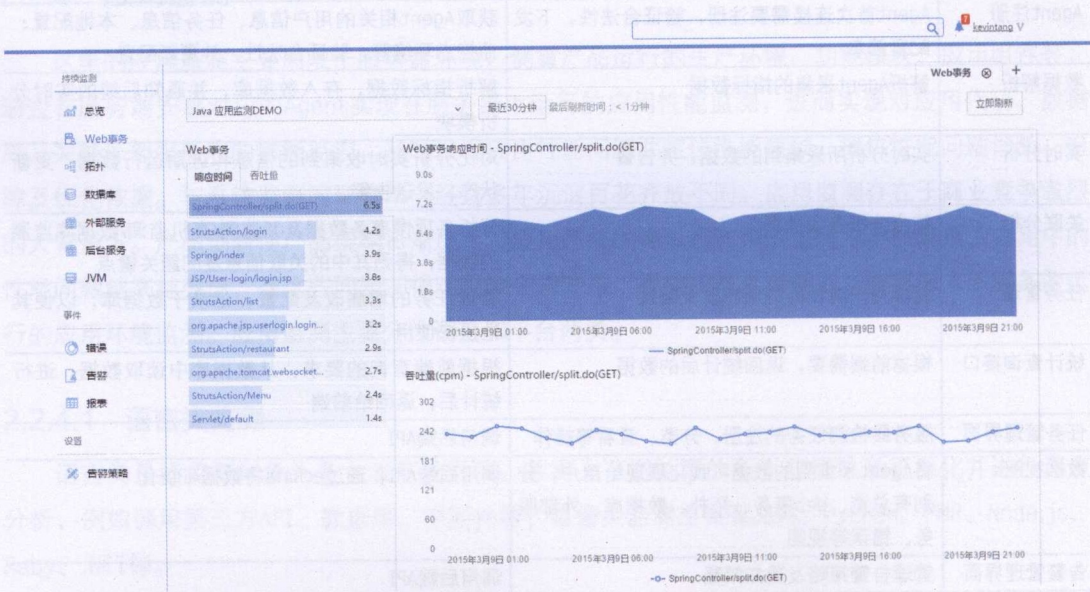


图2-32 语言类监测视图

2.2.4.1.1 Java监测

Java作为一种服务器端语言，面向企业级后端开发。2016年全球1月编程语言排行榜中，Java成为2015年度最受欢迎语言，如图2-33所示。Java跨平台性也受到更多开发者的青睐，因Java程序编译之后的代码不是能被硬件系统直接运行的代码，而是一种字节码，不同的硬件平台上安装有不同的Java虚拟机(JVM)，由JVM来把字节码再“翻译”成所对应的硬件平台能够执行的代码。因此对于Java编程者来说，不需要考虑硬件平台是什么。据JetBrains 2015年4月统计估算，全球有900万Java开发者，据Java官网显示美国有89%的桌面（或计算机）运行Java，Java是排名第一的部署平台，另外1.25亿台TV设备也在运行Java。特别是在开发语言日益丰富和多元化的当下，Java已经凭借着自身经久不衰的人气证明其经得住时间的严苛考验，在Java如此受欢迎的背景下，Java应用性能监测与优化具有重要意义。

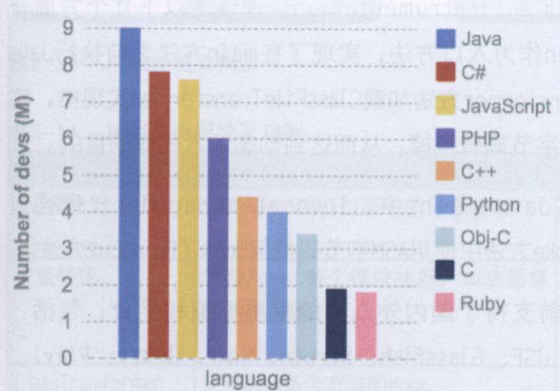


图2-33 全球语言开发者数量统计

Java监测价值及目标

- 1 全方位Java应用性能监测，便于管理整个应用架构，Java Agent可以监视各种应用程序性能指标和服务器参数，为整个应用架构提供统一的视图，探知Java应用之间的相互调用关系。
- 2 提供直观的Java应用诊断功能，准确定位故障。通过代码检测、监控记录或者参考网络/硬件等技术，帮助产品线对应用程序性能做出分析，将可能影响到用户的问题扼杀在萌芽状态。Java Agent通过内存快照，隔离所有与内存问题有关的详细信息，包括对象相互依赖性，帮助用户解决内存溢出、内存泄露等问题。Java Agent通过线程转储，查找死锁、闲置或繁忙线程池，帮助用户解决线程问题。
- 3 提供对Java应用关键业务的性能监测，对关键业务（如交易业务，如果交易数下降了，判断是否是故障影响）进行监测、分析，帮助了解业务性能和用户行为。Web事务监视帮助

监视从URL到SQL整个端到端的Web事务。可以监视由URL执行的Web组件、EJB、Java和SQL语句的性能规格。此外，为了识别性能瓶颈，还可跟踪各种J2EE和Java组件的各种方法。Agent通过历史method堆栈，实现Web事务的轨迹追踪，帮助用户追踪影响性能问题的主要因素。

- 4 个性化报表定制，为所有的Java应用重要参数提供了内置报表，可通过简单拖曳，实现自定义的个性化报表。

Java监测实现原理

- 1 Java监测通过Instrumentation代理及InvocationHandler动态代理，使用ASM字节码框架在字节码层面Hook了Java常用框架的关键方法。
 - 1) Instrumentation代理在Java Agent中，Instrumentation代理涉及以下几个方面，Instrumentation代理指定了premain作为入口方法，实现了在main方法之前执行Java Agent。Instrumentation通过addTransformer方法加载ClassFileTransformer实现类，实现了在class被装载到JVM之前将class字节码转换掉，从而达到动态注入代码的目的。
 - 2) InvocationHandler动态代理，在Java Agent中，InvocationHandler代理通过AgentWrapper触发invoke方法，在invoke方法中使用ASM字节码框架Hook了PointCut方法。
- 2 支持多框架和多平台，Java Agent目前支持了国内外众多的常用框架和平台，包括：Tomcat、CXF、Resin、JBoss、Spring、JSF、Glassfish、Struts、Solr、Jetty、Play1、TomEE、Grails、Hibernate、EJB、WebLogic、WebSphere、JMS。
- 3 支持多种数据库监测，Java Agent提供多种数据库监视，主动通知用户关于可能损害数据库性能的潜在问题，记录Java应用连接到的数据库资源、监视数据库增删改查操作，并且对数据库系统属性设定阈值，超出阈值时，系统将通过警报发出通知。Java Agent目前支持了常用的数据库，包括：MySQL、SQL Server、Postgres、Apache Derby、Oracle。新增了对NoSQL数据库的支持，包括：Memcache、MongoDB、Redis。
- 4 支持多种NoSQL数据库监测，Java Agent通过Hook了Memcache客户端（java_memcache、spymemcached、xmemcached）、MongoDB客户端（mongodb-jav0a-driver）及Redis客户端（jedis）的连接数据库和执行数据库的关键方法，实现了对NoSQL数据库的监测，帮助用户了解NoSQL数据库的性能问题。

Java监测架构及功能说明如图2-34和表2-11所示。

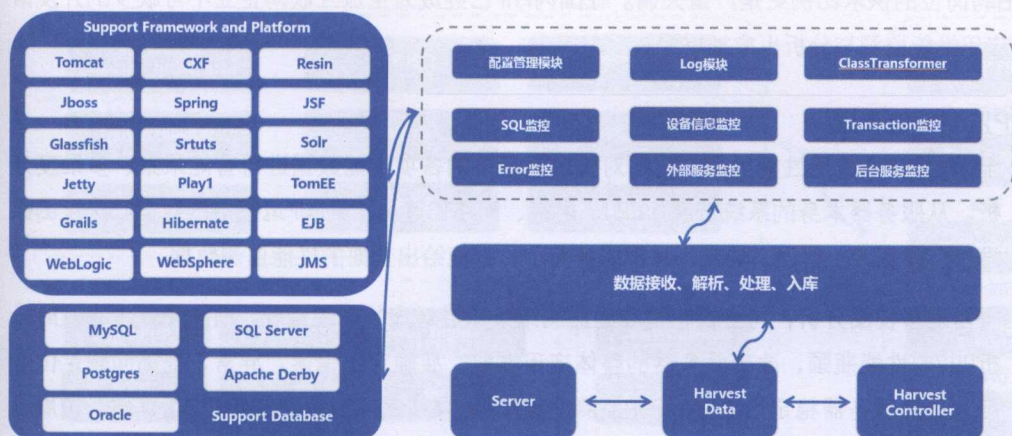


图2-34 Java监测架构图

表2-11 Java监测模块及功能

模块	作用	实现原理
Log模块	打印Agent运行过程中的日志信息	通过Info, Verbose, Debug, Warn, Error等级别, 控制日志信息的输出
配置管理	管理Agent各个模块功能, 如功能模块的开关等	通过Configuration单例记录各个模块的配置, Service启动时, Configuration为默认配置, 通过与后台交互, 可返回个性化配置
ClassTramformer	实现Java各个方法的Hook	通过Point Cut, Instrument, ASM, 实现了对各个方法的监控
Error监控	监控Server发生Error	通过监控Exception类实现对于Errors的监测
SQL监控	监控Server运行过程中的SQL调用	通过监控Java框架SQL执行方法, 实现对于SQL的监测
外部服务监控	监控Server运行过程中的外部服务	通过监控HTTPConnection方法, 实现对于外部服务的监测
后台服务监控	监控Server运行过程中的后台服务	通过监控后台线程方法, 实现对于外部服务的监测
设备信息监控	监控Server运行过程中的设备信息、内存/CPU信息等	调用了系统Framework提供的接口, 获取设备的基本信息, 内存/CPU等
Transaction监控	监控Server运行过程中的Web事务	通过监控Tomcat等Web服务器, 实现对于外部服务的监测
HarvestController	周期性将采集的数据发送给Server	Timer+Connection, 通过定时器周期性地采集的数据发送给Server

2.2.4.1.2 PHP监测

PHP是一种HTML内嵌式的语言, 语言的风格有类似于C语言, 是一门常用于Web编程的语言。2015年底, 自由UK Web对来自GitHub、RedMonk等4个网站、多种语言进行了统计, 综合排名第3。调查显示, 60%以上使用PHP技术。根据最新的Alexa TOP500中国网站排名统计, 有394家使用了PHP

技术，比例达到78.8%。随着IT业和互联网的高速发展，企业对PHP程序员的需求也大量增加，PHP程序员和招聘岗位的供求比例更是严重失调。在国内PHP已经成为主流互联网企业不可缺少的开发语言，PHP应用性能监测与分析也愈加重要。

PHP监测价值及目标

- 1 全面监控PHP应用性能监测，能够对脚本运行中的各项性能数据进行智能采集、多维度分析。从服务器本身的系统资源（CPU、内存、网络IO等），到PHP运行中的数据收集（函数堆栈、数据库、错误、异常、外部服务等），都能给出全面的性能监测数据。
- 2 丰富的多视图分析，将全面、精准的监测结果通过对应的视图呈现，有利于开发者及时分析PHP的性能瓶颈，改善服务器的整体应用性能。准确定位错误、异常，提高问题定位速度、提升服务器稳定性。同时PHP监测使用PHP扩展技术实现（通常使用C语言作为扩展语言），其中部分引入了C++的语言特性（类、STL相关），及较为成熟的库（日志模块、Socket通信模块等），从而提高监测本身的稳定性、安全性。
- 3 定制化的监测需求，除了全面的监测功能（函数调用堆栈、错误异常监控、系统资源监控），PHP监测根据用户自身需求，提供了对应的配置文件，产品线可以根据自身应用特点及需求，定制化性能数据的上报、告警。

PHP监测实现原理

- 1 PHP Agent执行流程主要由PHP内核、Zend引擎、扩展模块组成。PHP内核用来处理请求、文件流、错误处理等相关操作。Zend引擎用来将源文件转换成机器语言，然后在虚拟机上运行它。扩展层：一组函数、库类和流，PHP使用它们来执行一些特定的操作。
- 2 运行模式全面支持，PHP监测在运行时，能够感知不同的运行模式（CGI模式、多进程模式、多线程模式、FastCGI模式等）。对于不同的运行模式，PHP监测运用不同技术（共享内存、线程池等）进行数据采集。用户不需要知道PHP的运行模式，即可以轻松使用。
- 3 支持多种NoSQL数据库监测，PHP监测能够Hook到NoSql的关键API，实时跟踪PHP运行时的性能延时，帮助用户发现耗时的函数、语句。
- 4 支持PHP主流框架监测，支持PHP的不同框架、运行模式，无论使用的是哪一种主流PHP框架，都能快速部署使用和监测应用运行状态。

PHP监测架构及分析如图2-35和表2-12所示。



图2-35 PHP监测架构图

表2-12 PHP监测模块及功能

模块	作用	实现原理
Web事务监测	获取事务的函数名、事务总耗时、慢事务的堆栈信息	替换zend的函数回调（zend_execute），实现用户自定义函数的捕获
错误、异常监测	捕获PHP执行的错误、异常信息	替换zend的错误回调（zend_error_cb），实现错误、异常信息的捕获
数据库监测	实现对MySQL、Memcached、Redis、MongoDB中关键函数的监测	Hook数据库API，捕获操作语句、数据库调用耗时
外部服务	curl_exec、fsockopen等外部服务监测	Hook外部服务API，捕获外部服务的访问URL，及其耗时
服务器信息	获取PHP参数信息、系统信息	PHP守护进程实时统计操作系统信息
日志输出	打印Agent运行过程中的输出信息	通过配置文件，控制日志输出信息
慢事务阈值配置	指定事务函数堆栈上报的阈值	通过记录Server运行中慢事务耗时，对满足阈值的函数堆栈统计上报
慢SQL阈值配置	指定慢SQL上报的阈值	通过监控Server运行中SQL语句耗时，对慢SQL的语句信息统计上报
Transaction监控	监控Server运行中的用户自定义函数	重写zend的自定义回调函数（zend_execute），实现用户自定义函数的捕获
Error监控	监控Server运行中的错误信息	重写zend的错误回调函数（zend_error_cb），实现错误信息的捕获
Exception监控	监控Server运行中用户的异常信息	重写zend的异常回调函数（zend_throw_exception_hook），实现异常信息的捕获
SQL/NoSQL监控	监控Server运行中的SQL关键函数调用	Hook数据库API，获取操作语句、调用耗时
外部服务监控	监控Server运行中外部服务函数调用	Hook外部服务API，获取外部服务的访问URL、调用耗时
Server监控	获取PHP配置信息、Server系统信息	通过Agent获取PHP参数、Server硬件信息

2.2.4.1.3 .NET监测

根据Netcraft 2015年5月份的统计数据，在全球的Web服务器中，Microsoft的IIS，以236,288,843台的规模，占领27.83%的市场份额，而绝大部分的IIS服务器均选择了.NET作为其后台框架。另外，从2001年Mono框架的推出，到2014年Microsoft将.NET开源，.NET还在不断向其他操作系统渗透。对于开发者而言，.NET作为一项基础框架，支持多种语言开发，包括C#，VB.NET，C++，NET，J#，F#等，在TIOBE开发语言排行榜（2016年1月）中，C#排第4，VB.NET排第7，C++排第3，开发语言选择性大，且皆为市场主流。

同时国内的培训机构，为市场输送了大量.NET从业人员。由于.NET秉承了微软技术入门简单的特点，入门级开发者很多，资深开发者较为稀缺，两极分化严重。市场规模大，开发语言选择较多，开发人员水平参差不齐，这些特点造成人们在主观上担忧起.NET应用的性能表现，因此对.NET应用进行性能跟踪、优化的意愿更加强烈。

.NET监测价值及目标

- 1 全方位.NET应用性能监测，针对Web请求、数据库调用、外部服务、后台任务等，监测采集每一次执行，统计分析响应性能和吞吐量，简洁明了地展示各项指标趋势。分析耗时最长、吞吐量最大的访问，详细了解应用程序健康程度，详细展示各条慢服务的执行细节，还原调用堆栈，突出耗时代码，轻松做到软件性能优化。
- 2 全面监控运行过程中的异常错误，还原出错位置，统计重现频率，针对重大问题防患于未然，大幅提升程序稳定性。关联监测数据库和外部调用，合并分析服务端应用的受影响因素，快速定位软件框架中的待优化模块，提升整体性能。
- 3 详细、个性化的配置，天下没有两片完全一样的树叶，没有两个应用会完全一样，通过多角度、详尽的配置清单，可以自由控制各类阈值、过滤项等。甚至针对同一应用运行的不同服务器，均可以设置不同的标准，专注于自身关心的问题，排除干扰。
- 4 及时、多样化的告警，在用户体验指数过低、错误率过高或者持续一段时间接收不到数据时，.NET监测服务会第一时间发出告警。后续支持配置监测某一条或多条Web请求，当其响应时间过长或者吞吐量异常时，立即发出告警。

.NET监测实现原理

- 1 为了方便开发人员对.NET应用进行性能分析，微软提供了一套Profiling API机制，其能深入了解.NET应用的内部运行情况。.NET监测采集端的核心模块编写为COM组件，并根据Profiling API的要求，实现了IcorProfilerCallback2接口。通过改写注册表，实现了根据

.NET应用一起启动并完成注入。IcorProfilerCallback2在.NET应用加载Assembly、Module及执行Function时，均会进行回调，经过过滤，相当于Hook了一系列的函数。同时.NET采集端根据.NET Framework的event通知、异常捕捉、系统信息获取接口等机制，获取更加全面的监测数据。

- 2 多语言支持，.NET监测服务根据.NET基础框架进行监测，得益于.NET技术的多开发语言特性，无论采用的是C#还是VB.NET、C++.NET或者其他，只要基于.NET Framework，均可进行监控。
- 3 支持多种框架，目前已经支持的框架包括ASP.NET MVC，ASP.NET Web API，SOAP-based Web Services，WCF等。
- 4 支持多种数据库，.NET监测并非直接监测数据库进程，而是从Web请求处理和后台任务的代码中，自动提取数据库调用相关信息。目前已经支持的数据库包括MS SQL Server，Oracle，MySQL，PostgreSQL，MongoDB，Redis等。

.NET监测架构及说明如图2-36和表2-13所示。

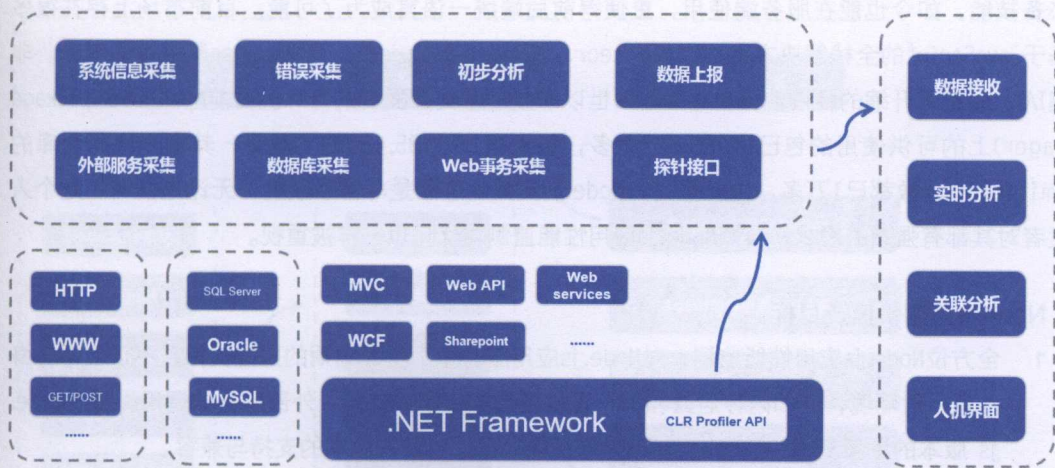


图2-36 .NET监测架构图

表2-13 .NET监测模块及功能

模块	作用	实现原理
探针接口	跟随.NET应用启动	采用CLR Profiler API，实现并注册COM组件，在.NET应用启动时，将Agent跟随启动
Web事务采集	采集Web访问.NET应用时的各项指标	在各框架的入口进行注册，收集每次Web请求的各项数据，支持MVC、Web API、Web Services、WCF、Sharepoint等框架

续表

模块	作用	实现原理
数据库访问采集	采集.NET应用访问数据时的各项指标	在数据库访问时，被动收集相关信息，支持SQL Server、Oracle、MySQL、MongoDB等
外部访问采集	采集.NET应用访问第三方接口时的各项指标	在访问第三方HTTP接口时，被动收集相关信息
错误采集	采集.NET运行过程中内部异常信息	捕捉.NET的异常，收集发生异常时的相关信息
系统信息采集	采集运行过程中CPU、内存、线程、垃圾回收等各项信息	综合采用.NET库函数及Profiler API，针对各项指标分别采集
数据上报	上报Agent采集的数据	汇总采集的数据，并定时整合发送给服务器
数据实时分析	实时分析所采集到的数据，并告警	对比分析实时收集到的信息和近期运行数据，更新状态，发送告警
数据关联分析	综合分析各项数据	分析各项慢事务数据及JS监测/真机监测/数据库监测的数据，得到其中的关联信息及问题关键点

2.2.4.1.4 Node.js监测

借助于Google的V8引擎使得JavaScript在服务端运行成为可能，JavaScript作为前端工程师的基本必备技能，如今也能在服务端使用，更使得前后端统一语言成为了可能。目前市场上存在很多的基于JavaScript的全栈解决方案，比如Meteor、MEAN、Tower.js等。而Node.js 基于事件驱动、非阻塞I/O、轻量可升缩的特性使得自2009年面世以来就非常受开发者的青睐。目前npm(node package manager)上的可供使用的包已达225,527之多，每天有127,985,436的下载量，其官方代码仓库的commit数和star数都已1万多，由此可见，Node.js及其生态圈是非常活跃的，无论是公司还是个人开发者对其都有强烈的需求，自然Node.js应用性能监测与优化也一样被重视。

Node.js监测价值及目标

- 1 全方位Node.js应用性能监测，对Node.js应用服务整个生命周期的监控支持，不仅包括自身的运行时环境，例如内存泄露和GC，还包括数据库事务处理、外部服务的调用。紧跟Node.js 版本的更新及相关的Web框架组件的升级，保证对最新版本的支持与兼容。
- 2 部署使用简单，对使用者的服务资源无损耗，监控探针通过hook的方式采集数据，不需要修改源代码，实时上报应用的各项指标数据，产品线通过简单的Web页面就能够对应用进行监控以及分析。
- 3 自定义监测，对于Agent自身没有涉及的监控指标，可由开发者自定义监控插件，采集对应的指标数据，并按照规定的数据格式上报，由服务器分析处理可视化以及告警。

Node.js监测实现原理

- 1 Node.js监测采用的是无代码侵入的方式，不需要在业务中修改任何代码，仅需要在APP入口的文件的顶部加入简单配置即可享用对Node Server的监测。Node Agent 主要是通过对Node API进行hook的方式采集应用数据，另外再分别对不同的应用框架的特性做适配，支持扩展能很好地支持目前市面上的绝大部分Node应用。Agent在启动的时候对核心方法进行Hook，主要包括http、process、time、net、crypto、fs等核心模块，在APP运行时collector模块按照sample定期采集应用数据，进行预处理后上传到分析处理服务，提供可视化页面。
- 2 支持主流框架，对常见的应用框架Express、Restify、Koa有全面的支持。
- 3 支持主流数据库，除了传统的关系型数据库外，结合Node.js自身的特点，对非关系型的数据库也提供了全面的支持，比如MongoDB，另对Redis、Memcached也提供了全面的支持。
- 4 多种运行环境的支持，在生产环境中大多开发者都是采用PM2 forever等对Node.js应用进行管理，对此种环境下也有非常好的支持。

Node.js监测架构及说明如图2-37和表2-14所示。



图2-37 Node.js监测架构图

表2-14 Node监测模块及功能

模块	作用	实现原理
Agent启动	加载license_key及相关配置到Agent中，并在服务端注册该NodeServer	通过配置的license_key到服务器上注册该APP，并更新默认配置
patchModule	为Module打补丁，方便对信息的抓取	重写module的_load方法，添加数据记录

续表

模块	作用	实现原理
collector	采集及分析采集到的数据进行上报到存储模块，供分析处理&持久化	通过默认配置和用户自定义配置，抓取对应模块的数据
样本sampler	样本采集模版，采集的指标以及上报的数据模版	汇总采集的数据，并定时整合发送给服务器
Agent数据上报	获取Node Server服务器的各项指标数据，上传到集中存储模块供服务端进行数据分析	将collector根据sampler采集到的数据按照配置文件的频率上报给数据存储模块供服务分析

2.2.4.1.5 Ruby监测

2015年，据JetBrains统计估算，全球约有180万Ruby开发者。RedMonk 2015年第三季度编程语言排行榜中，Ruby与C#、C++并列排行第五。Ruby最常用于后端开发，目前已有不少成功案例 Airbnb、Shopify、Bloomberg、Hulu、Slideshare及更多热门网站就是用Ruby on Rails构建的。2015年8月统计各种语言在Github的库存数，得出了Top100全球最受欢迎的编程语言，Ruby以155.9万个排名第二。RubyGems.org官方统计，截至2016年1月，托管到RubyGems的Ruby应用数量为11.2万个。可以预见Ruby仍有较大上升空间，对Ruby应用性能的监测与优化，需要尽早重视。

Ruby监测价值及目标

- 1 全面的Ruby应用性能监测，涵盖网络状态、Web响应情况、设备硬件、数据库情况等性能数据。并提供丰富的分析视图功能，对Ruby应用性能做出全面评估，提高Ruby应用的可靠性和质量。
- 2 瓶颈问题深度追踪，瓶颈问题快速定位。通过丰富的视图，可快速定位应用的健康程度和性能状态，可以追溯至详细信息，包括地理信息、关键事务、错误统计等性能度量点，快速定位性能问题。Ruby Agent采用了alias_method的方式，Hook了Ruby常用框架的method，实现关键事务的深度性能剖析、关键事务性能分析、事务Traces记录查看、代码级别的深度性能问题追溯。
- 3 关键事务分析，对于用户需要监测的重点业务，Ruby Agent提供“关键事务分析”功能，以满足个性化需求。Ruby Agent对设定的关键事务进行详细分析，记录其吞吐量、响应情况、错误情况、Trace等，帮助了解关键业务的性能问题（如交易业务，发生吞吐量下降，则判断是否应用故障，若是，则追踪详细原因）。
- 4 错误代码级分析定位，错误定位到代码行，Ruby Agent通过Hook Exception类的方式，截取Ruby应用的异常信息，追踪错误发生时的应用情况，记录错误出现的频率和详细信息。错误追踪可定位问题到具体的代码行。Ruby Agent还可以通过过去错误堆栈信息，追踪错误轨迹。

- 5 开放性API, Ruby Agent提供开放性API, 可根据API文档的说明, 实现自定义的监测功能。并为Ruby应用重要参数提供了内置报表, 可通过简单拖曳, 实现自定义的个性化报表。

Ruby监测实现原理

- 1 Ruby 监测通过alias_method、pub-sub机制及Rack多种方式Hook了Ruby框架并监测相关的数据。
- 1) alias_method是给方法起一个别名。方法别名定义后, 即使对应的方法在后面的代码中重新定义(即修改内部实现)后, 别名仍然可以调用到修改前的方法。
- 2) pub-sub是publish-subscribe两个单词的缩写, 其意思就是发布-订阅, 它提供了一种事件处理函数的注册方式。采用pub-sub机制探针只需要订阅特定类型的消息, 然后进行数据再加工。
- 3) Rack起源于Python的WSGI协议, 是一个语言相关的HTTP服务端接口(CGI, FCGI, SCGI 是语言无关的接口, WSGI, Rack, Servlet 则属于语言相关的接口)。
- 2 主流框架和平台的支持, Ruby Agent目前支持了Rails及Sinatra两大主流的平台, 同时支持Resque, Sidekiq 及 Delayed Job框架。
- 3 主流数据库的支持, Ruby Agent目前支持了主流的数据库, 包括: MySQL、SQL Server、Postgres、Apache Derby、Oracle。

Ruby监测架构及说明如图2-36和表2-15所示。

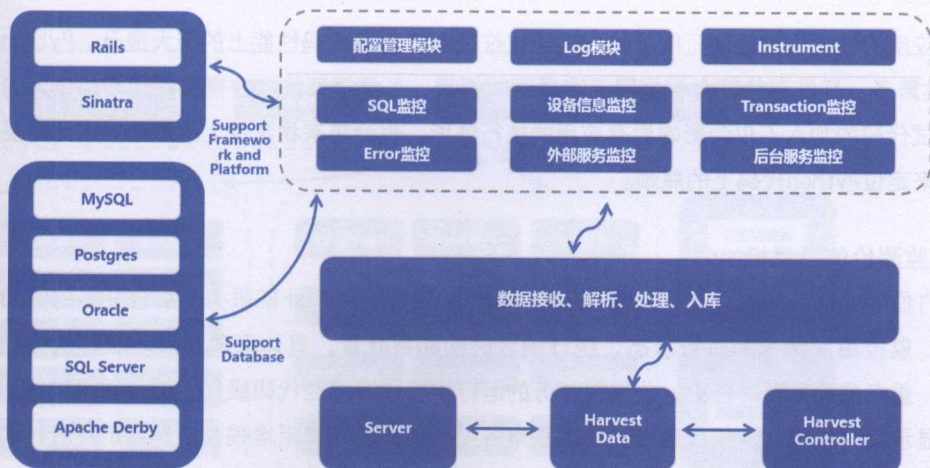


图2-38 Ruby监测架构图

表2-15 Ruby监测模块及功能

模块	作用	实现原理
Log模块	打印Agent运行过程中的日志信息	通过Info, Verbose, Debug, Warn, Error等级别, 控制日志信息的输出
配置管理	管理Agent各个模块功能, 如功能模块的开关等	通过Configuration单例记录各个模块的配置, Service启动时, Configuration为默认配置, 通过与后台交互, 可返回个性化配置
Instrument	实现Ruby 各个方法的Hook	通过alias_method实现了对各个方法的监控
Error监控	监控Server发生Error	通过监控Exception类实现对于Error的监测
SQL监控	监控Server运行过程中的SQL调用	通过监控Ruby框架SQL执行方法, 实现对于SQL的监测
外部服务监控	监控Server运行过程中的外部服务	通过监控HTTPConnection方法, 实现对于外部服务的监测
后台服务监控	监控Server运行过程中的后台服务	通过监控后台线程方法, 实现对于外部服务的监测
设备信息监控	监控Server运行过程中的设备信息, 内存/CPU信息等	调用了系统Framework提供的接口, 获取设备的基本信息, 内存/CPU等
Transaction监控	监控Server运行过程中的Web事务	通过监控Tomcat等Web服务器, 实现对于外部服务监测
HarvestController	周期性将采集的数据发送给Server	Timer+Connection, 通过定时器周期性地将采集的数据发送给Server

2.2.4.1.6 Python监测

Python因为其庞大的第三方库而广受欢迎, 现在有越来越多的人将Python作为网站的后台支撑语言。国外的诸如Google、DropBox、Instagram、NASA等, 国内有豆瓣、知乎、美团等网站, 均采用了Python作为其后台编程语言。Python的常用Web框架就多达20余种, 可以满足各类产品要求。在TIOBE开发语言排行榜(2016年1月)中, Python排在第5, 并呈逐年上升的趋势。从2007年以来, Python已经两次荣获TIOBE年度开发语言。根据JetBrains的估算, 全球约有400万的Python开发人员。

Python应用的性能相对较弱, 简单的代码优化往往可以带来代码性能上的极大提升。Python的Web框架种类繁多, 开发者往往会根据需要进行选择不同框架, 各类框架的监控需要额外增加很大的人力成本。调试代码的加入不仅会影响原有应用的执行性能, 也会带来稳定性上的风险。因此需要简单高效监测来定位Python代码上的瓶颈。

Python监测价值及目标

- 1 全方位Python应用性能监测, 采集Web请求、数据库访问、外部服务、后台任务的详细信息, 监控每次请求的执行状态, 统计响应时间和吞吐量, 具体形象地展示程序的性能走势。重点监控最耗时任务, 收集慢任务的运行情况, 详细至代码级别, 记录调用堆栈, 高亮显示耗时代码, 从代码层面上精确采集各项指标, 还原调用堆栈, 准确展示耗时代码, 轻松定位应用程序的性能瓶颈。并且准确地给出出错记录, 采集每次异常信息, 统计出错频率, 展示错误率走势。

- 2 轻松集成，Python监测服务充分利用Python的动态语言特性，通过独立的代码模块，对现有框架函数进行封装替换，在不需要修改原有代码的基础上，支持对任何函数进行捕获。另外具有独立的安装形态，Agent为一个普通的第三方库，打包成通用的egg文件，通过pip或者easy_install命令，无论在线还是离线，一行指令即可轻松安装。
- 3 个性化的配置，详细的配置清单，可以区别对待各应用，甚至分布式部署于不同服务器上的同一个应用，排除不必要的干扰，根据各应用的复杂度和各服务器的性能，灵活改进各模块性能。

Python监测实现原理

- 1 Python监测端采用了和Decorator类似的机制。Decorator通过@进行函数修饰，可以非常方便地修改原函数，在其执行前或执行后增加代码。在Python语言中，函数也是变量，因此，可以通过闭包生成一个全新函数，来替代原函数。全新函数在执行原函数的同时，可以执行任意代码。得益于Python的动态语言特性，后续代码在调用原函数的位置，会改为调用新函数。基于该种机制，可以hook一系列的函数，完成对Web框架、数据库调用等信息的采集。
- 2 多框架支持，目前已经支持多种的Web框架，包括：bottle, cherrypy, django, flask, pylons, pyramid, tornado, web2py等。同时，支持快速新增框架或者用户自行开发的框架。
- 3 多数据库支持，目前已经支持多种数据库软件，包括Oracle, MySQL, PostgreSQL, SQL Server, MongoDB, Redis等。

Python监测架构及说明如图2-39和表2-16所示。

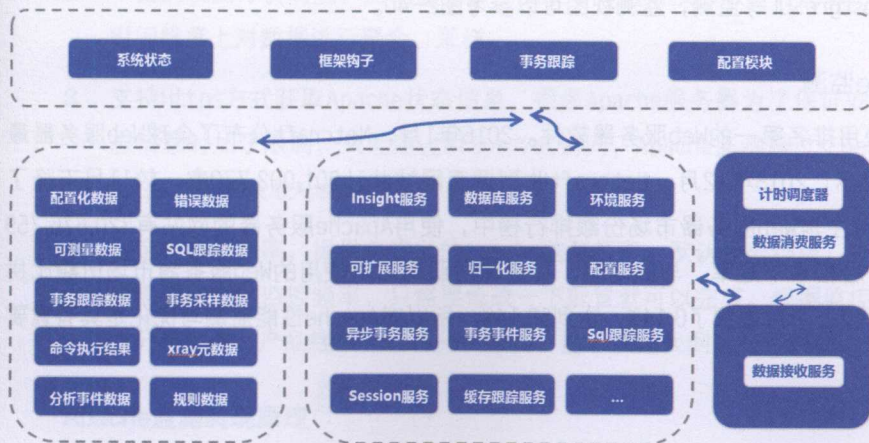


图2-39 Python监测架构图

表2-16 Python监测模块及功能

作用	实现原理
采集系统性能指标趋势	通过周期性地采集系统设备状态来获取cpu/内存/io等指标
跟踪tornado/django框架方法调用栈	使用python动态获取属性或方法getattr(),setattr(),delattr(),hasattr()来动态跟踪到框架指定函数,然后使用自定义的wrap方法实现目标方法的劫持
跟踪Web事务中内部、外部方法调用栈	事务跟踪中会使用Insight服务、数据库跟踪服务、SQL跟踪服务等,主要使用Python的with语句实现事务中内外部服务接口的函数级跟踪;比如:TimeTrace会在_enter_中记录transaction活跃节点的push时间戳,在_exit_中记录transaction结束时间戳
打印Agent运行过程中的日志信息	通过info/verbose/debug/warn/error等级别,控制日志信息输出
管理Agent各个模块功能开关	使用单例设计模式,service启动时发送connect数据到服务器,服务器反馈定制化模块配置
实现Python各个方法的hook	使用Python动态获取属性或方法getattr(),setattr(),delattr(),hasattr()来动态跟踪到框架指定函数
监控server发生Error	使用Python的traceback来进行监测
监控server运行过程中的SQL调用	通过监控Python框架的SQL执行方法实现对SQL监测
监控server运行过程中的后台服务	通过监控后台线程方法实现对于后台服务的监测
监控server运行过程中的设备信息、系统状态	通过调用周期性采集系统设备文件接口来获取相关状态数据
监控server运行过程中的Web事务	主要使用Python的with语句实现事务中内外部服务接口的函数级跟踪
周期性采集数据发送给服务器	使用定时器+消费者生产者模型实现

2.2.4.2 平台类监测

平台类是指监测对象主要为日益丰富和开源的优秀第三方应用服务,并将上下流关联的所有应用性能可视化并进行深度分析,平台类监测主要有Nginx、Redis、Apache、Tomcat、Memcached、MongoDB、MySQL、PostgreSQL等监测,监测视图可以参考图2-40。

2.2.4.2.1 Apache监测

Apache是世界使用排名第一的Web服务器软件。2016年1月,Netcraft公布了全球Web服务器最新数据。根据数据显示,2015年12月,Netcraft收到调查网站共计901,002,770家,较11月下降了0.22%。其中,在全球主流Web服务器市场份额排行榜中,使用Apache服务器的网站有320,676,759家,份额为35.59%,轻松拿下冠军。与此同时,在全球活跃网站所使用的Web服务器市场份额比拼中,也是稳居第一,较上个月增加了0.14%,达到50.14%。所以对Apache性能监测与优化是具有重要意义的。

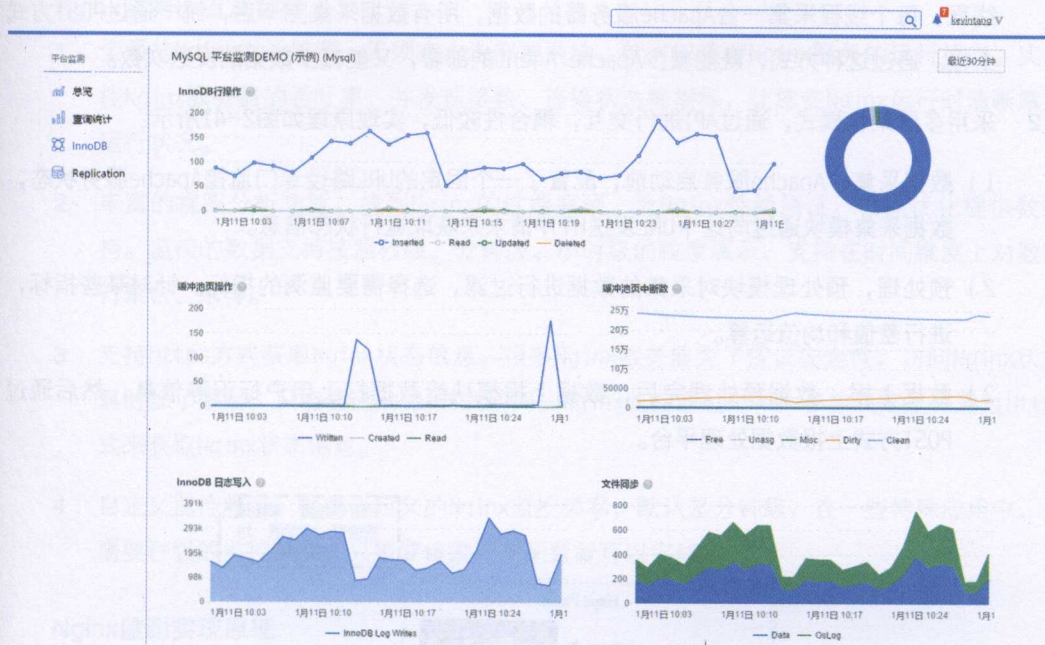


图2-40 平台类监测视图

Apache监测价值及目标

- 1 全面的Apache性能监测，无须在本地部署系统，就能监控Apache服务器的运行状态。实时监控Apache服务器的吞吐率、启动的线程数、消耗的系统资源等指标，清晰掌握Apache服务器运行状况。
- 2 丰富的视图分析功能，监测数据多粒度展示并支持按秒级、分钟级、小时级展示，支持在时间维度上对数据进行聚合、采样。
- 3 支持Https方式获取Apache状态信息，很多Apache服务器为了保证安全性，访问Apache状态信息时加了限制，只能通过Https方式访问，Apache监测通过scheme参数决定是否通过Https方式来获取Apache状态信息。
- 4 自定义监测频率，提供自定义的Apache监测频率，默认是分钟级，在一些特殊应用中，如果需要秒级的监控频率，只需要修改一下配置就可以完成。根据监控的数据，自动生成运行分析报告，产品线根据运行分析报告，能够快速找到运行过程中的一些问题。

Apache监测实现原理

- 1 多线程方式进行数据采集，单个Apache Agent可以监控多个Apache服务器，后台启动多个

线程，每个线程采集一台Apache服务器的数据，所有数据采集完毕后，统一通过POST方式上报。通过这种方式，既能减少Apache Agent的部署，又能减少数据的发送次数。

2 采用多层架构模式，通过API进行交互，耦合性较低，实现原理如图2-41所示：

- 1) 数据采集，Apache服务启动前，配置了一个固定的URL路径专门监控Apache服务状态，数据采集模块通过向这个URL发送HTTP请求来获取运行状态信息。
- 2) 预处理，预处理模块对采集的数据进行过滤，选择需要监测的指标。针对某些指标，进行差值和均值运算。
- 3) 数据上报，数据预处理完后，数据上报模块给数据打上用户标识等信息，然后通过POST方式上报数据处理平台。

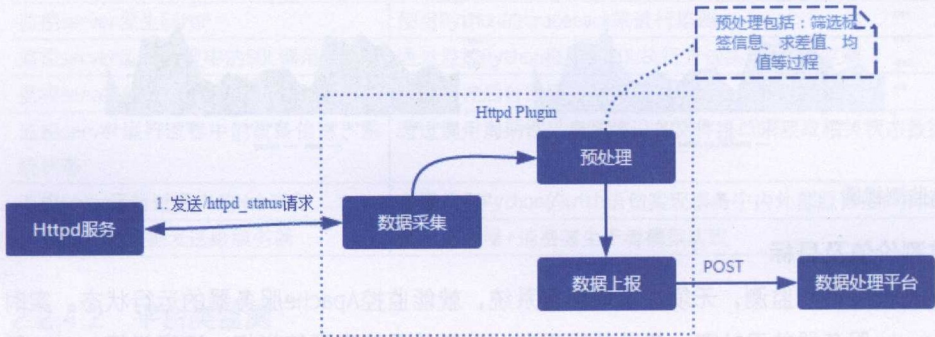


图2-41 Apache监测原理

- 3 配置灵活，所有的参数都是通过YAML文件进行配置的，并且提供了一个默认值。如果需要修改参数，只需要修改这个配置文件，然后重新启动即可生效。
- 4 支持主流的Linux平台，Apache Agent采用Python语言开发，不依赖系统相关库，支持市面上主流的Linux发行版本。

2.2.4.2.2 Nginx监测

Nginx是一个高性能的服务器。据IDC统计，2015年6月，Nginx网络主机数明显增长，净增长量达到22800台，市场份额较同期增长了0.28%，达到了12.4%。Apache、IIS和Nginx加在一起占了全世界网络主机总数的88%以上，意味着这些厂商目前是网络主机市场最受欢迎的选择。从2015年6月的Web Server报告中可以看出，Nginx是三家中唯一一家市场份额持续增长的公司，相较于去年已经增长了三个百分点。Nginx在Alexa前一百万网站中的市场份额也在稳定地提高中，目前在前一百万网站中的占有率为21.9%，所以对Nginx的性能监测与优化对于企业而言是重要的工作。

Nginx监测价值及目标

- 1 全面的Nginx性能监测，无须在本地部署系统，就可以监控Nginx服务的运行状态。实时监控Nginx服务器的吞吐率、并发连接数、连接状态等指标，让您在Nginx运行时清晰掌握其运行状况。
- 2 丰富的视图分析功能，找到Nginx的性能瓶颈，为Nginx性能测试、性能优化提供数据支持。监控的数据支持按照秒级、分钟级、小时级的粒度展示，支持在时间维度上对数据进行聚合、采样。
- 3 支持Https方式获取Nginx状态信息，很多Nginx服务器为了保证安全性，访问Nginx状态信息时加了限制，只能通过Https方式访问。Nginx监控通过scheme参数决定是否通过Https方式来获取Nginx状态信息。
- 4 自定义监控频率，提供自定义的Nginx监控频率，默认是分钟级，在一些特殊应用中，如果需要秒级的监控频率，只需要修改一下配置就可以完成。

Nginx监测实现原理

- 1 多线程方式进行数据采集，单个Nginx Agent可以监控多个Nginx 服务器，后台启动多个线程，每个线程采集一台Nginx 服务器的数据，所有数据采集完毕后，统一通过POST方式上报。通过这种方式，减少Nginx Agent的部署。
- 2 采用多层架构模式，通过API进行交互，耦合性较低，实现原理如图2-42所示。

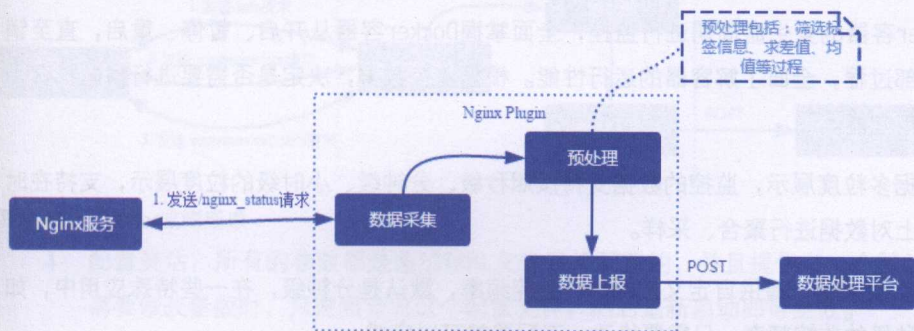


图2-42 Nginx监测原理

- 1) 数据采集，Nginx服务启动前，配置了一个固定的URL路径专门监控Nginx服务状态，数据采集模块通过向这个URL发送HTTP请求来获取运行状态信息。
- 2) 预处理，预处理模块对采集的数据进行过滤，选择需要监测的指标。针对某些指标，进行差值和均值运算。

3) 数据上报, 数据预处理完后, 数据上报模块给数据打上用户标识等信息, 然后通过 POST 方式上报数据处理平台。

- 3 配置灵活, 所有的参数都是通过YAML文件进行配置的, 并且提供了一个默认值。如果用户需要修改参数时, 只需要修改这个配置文件, 然后重新启动即可生效。
- 4 支持主流的Linux平台, Nginx Agent采用Python语言开发, 不依赖系统相关库, 支持市面上主流的Linux发行版本。

2.2.4.2.3 Docker监测

2015年, Docker实现了爆发式发展, 自Red Hat支持Docker起, Amazon、Google、VMware等云服务提供商以及主流的Linux提供商均开始支持Docker, IBM、MicroSoft相继宣布与Docker社区建立合作关系。Docker容器的下载量更是由年初的52万次增长到年末的1亿多次, 增长了接近190倍。基于Docker标准的应用程序数量达到7.1万, 较年初增长了12倍。Docker作为基于容器技术的轻量级虚拟化解决方案已经大势所趋, 而为了能够更精确地限定每个容器能使用的资源状况, 所以我们需要实时获取容器运行时使用资源的情况。

Docker监测价值及目标

- 1 全面的Docker性能监测, 监控Docker容器使用的内存、CPU、网络IO等资源, 还对Docker宿主机的CPU、内存、镜像数、容器数等指标进行监控, 全面了解宿主机上每一个Docker容器的资源消耗情况, 为性能分析和优化提供必要的数据支撑。
- 2 对Docker容器的全生命周期进行监控, 全面掌握Docker容器从开启、暂停、重启, 直至销毁的全部过程, 全面了解容器的运行性能。根据监控数据, 决定是否需要横向扩容, 保证应用性能。
- 3 监控数据多粒度展示, 监控的数据支持按照秒级、分钟级、小时级的粒度展示, 支持在时间维度上对数据进行聚合、采样。
- 4 自定义监控频率, 提供自定义的Docker监控频率, 默认是分钟级, 在一些特殊应用中, 如果需要秒级的监控频率, 只需要修改一下配置就可以完成。

Docker监测实现原理

- 1 Docker Agent通过Docker原生接口进行数据采集, 首先使用Docker Info接口来获取Docker系统信息, 这些信息包含了非常有用的数据, 如: 容器数、镜像数、CPU、总内存等。其次, 使用Docker Containers Info接口来取得容器的运行信息和容器标识, 容器名称, 正在

运行的容器有哪些。最后,通过Docker Containers Stats接口获取容器的运行性能指标,包括CPU使用率、内存指标、网络指标等。

- 2 多线程方式进行数据采集,单个Docker Agent可以监控集群中多个Docker宿主机,后台启动多个线程,每个线程采集一台Docker宿主机的数据,所有数据采集完毕后,统一通过POST方式上报。通过这种方式,减少Docker Agent的部署。
- 3 采用多层架构模式,通过API进行交互,耦合性较低,实现原理如图2-43所示。

- 1) 数据采集,数据采集模块通过Docker Remote API来获取Docker运行时的状态信息。首先通过/info请求API获取宿主机资源信息,然后通过/containers/json请求API获取Docker容器列表信息,最后遍历Docker容器列表,针对启动的Docker容器,通过/containers/(id)/stats请求API获取每个容器占用的资源信息。
- 2) 预处理,预处理模块对采集的数据进行过滤,选择需要监测的指标。针对某些指标,进行差值和均值运算。
- 3) 数据上报,数据预处理完后,数据上报模块给数据打上用户标识等信息,然后通过POST方式上报数据处理平台。

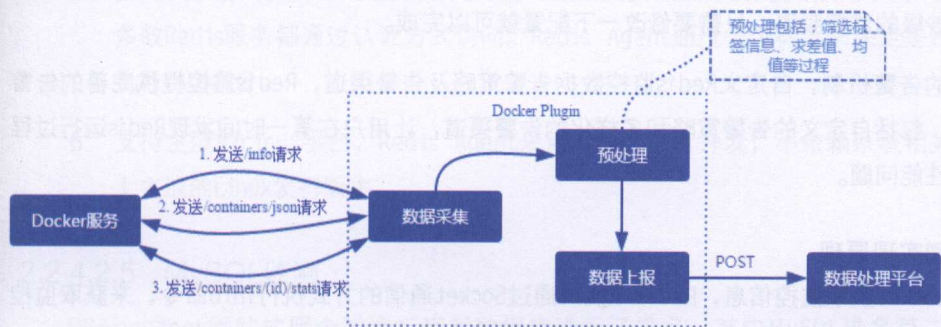


图2-43 Docker监测原理

- 4 配置灵活,所有的参数都是通过YAML文件进行配置的,并且提供了一个默认值。如果用户需要修改参数时,只需要修改这个配置文件,然后重新启动即可生效。
- 5 支持主流的Linux平台,Docker Agent采用Python语言开发,不依赖系统相关库,支持市面上主流的Linux发行版本。

2.2.4.2.4 Redis监测

Redis作为一种高性能键值数据库,通过提供多种键值数据类型来适应不同场景下的数据存

储需求。最近几年, Redis的企业客户数在不断增加, 从2014年的3400多家上升至2015年的5000多家。另外, 截止2015年12月底, Redis通过官网的下载量已达到7800万次。2015年11月, Redis Lab对Redis用户进行了随机调查。调查结果显示, 有71%的用户计划增加Redis节点, 应对应用规模的不断扩大。应用领域主要集中在社交、零售、金融、游戏方面, 这些领域的比例超过80%, 应用场景主要是把Redis作为消息缓存、消息队列, 两者的比例达到70%, Redis的性能监测与优化已经成为必选项。

Redis监测价值及目标

- 1 全面的Redis性能监测, 无须用户在本地部署系统, 就可以监控Redis服务的运行状态, 除了监控Redis服务使用的CPU、内存、网络IO等系统资源, 还从客户端连接情况、各个库包括Key的数量、主从同步等方面对Redis服务进行监控, 保证用户能够观察到每一个重要指标。
- 2 实时监控Redis的性能、消耗的资源、存储的数据量等指标, 同时也支持对指标的历史信息进行查询。监控数据多粒度展示, 监控的数据支持按照秒级、分钟级、小时级的粒度展示, 支持在时间维度上对数据进行聚合、采样。
- 3 自定义监控频率, 提供自定义的Redis监控频率, 默认是分钟级, 在一些特殊应用中, 如果需要秒级的监控频率, 只需要修改一下配置就可以完成。
- 4 完善的告警机制, 自定义Redis监控数据告警策略及告警渠道, Redis监控提供完善的告警机制, 包括自定义的告警策略和多样化的告警渠道, 让用户在第一时间发现Redis运行过程中的性能问题。

Redis监测实现原理

- 1 基于Socket获取监控信息, Redis Agent通过Socket通信的方式执行Info命令, 来获取监控信息。获取的信息全面, 而且不依赖于任何Redis客户端, 降低用户部署Redis Agent的复杂度。
- 2 多线程方式进行数据采集, 单个Redis Agent可以监控集群中多个Redis服务, 后台启动多个线程, 每个线程采集一个Redis服务的数据, 所有数据采集完毕后, 统一通过POST方式上报。通过这种方式, 减少Redis Agent的部署。
- 3 采用多层架构模式, 通过API进行交互, 耦合性较低, 实现原理如图2-44所示。
 - 1) 数据采集, 数据采集模块通过Socket与Redis服务建立连接, 然后执行info命令来获取Redis运行状态信息, 最后关闭Socket来释放连接。

- 2) 预处理, 预处理模块对采集的数据进行过滤, 选择需要监测的指标。针对某些指标, 进行差值和均值运算。
- 3) 数据上报, 数据预处理完后, 数据上报模块给数据打上用户标识等信息, 然后通过 POST 方式上报数据处理平台。

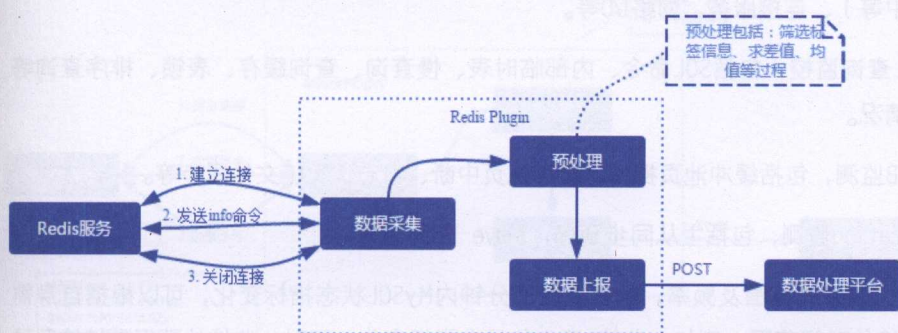


图2-44 Redis监测原理

- 4 配置灵活, 所有的参数都是通过YAML文件进行配置的, 并且提供了一个默认值。如果用户需要修改参数时, 只需要修改这个配置文件, 然后重新启动即可生效。
- 5 支持认证访问, Redis未授权访问可能导致远程获取服务器的权限, 为了解决这个问题, 大多数Redis服务都通过认证方式访问。Redis Agent通过一个标记来决定是否通过认证方式执行Info命令。
- 6 支持主流的Linux平台, Redis Agent采用Python语言开发, 不依赖系统相关库, 支持市面上主流的Linux发行版本。

2.2.4.2.5 MySQL监测

DB-engines网站按照全球流行度对数据库进行了排名, 其中MySQL排名第二, 成了仅次于Oracle的关系型数据库。2016年1月, DB-engines网站再次排名时, MySQL以1299分的成绩牢牢稳固第二位。而就2015年第三方市场调查机构 Evans 数据公司最近公布的一系列客户调查数据显示, 在过去两年里, MySQL在所有开发者使用的数据库中获得了25%的市场份额, Evans 公司的本次调查显示, 数据库的使用者中有40%是开发人员, 而两年前这一数据是32%。

MySQL越来越被企业级所接受, 但随着企业发展, MySQL存储数据日益膨胀, MySQL的性能分析、监控预警、容量扩展议题越来越多, 例如, MySQL资源消耗情况, 索引使用效率, 主从库Replication延迟, InnoDB死锁等问题, 尤其是线上的服务高度依赖数据库, 故而企业对MySQL的监控极为迫切。

MySQL监测价值及目标

- 1 全面的MySQL性能监测，仅需花费少量时间部署安装，且无须重启MySQL就能持续稳定的监控MySQL的运行。实时监控MySQL的各项运行指标，同时支持对指标的历史信息进行查询。
 - 1) MySQL性能监控，包括SQL查询频次、资源利用率（临时表，缓存使用率，InnoDB缓存池命中等）、连接服务、网络I/O等。
 - 2) MySQL查询监控，包括SQL命令、内部临时表、慢查询、查询缓存、表锁、排序查询等开销情况。
 - 3) InnoDB监测，包括缓冲池页操作、缓存池页中断、日志情况、文件同步等。
 - 4) Replication监测，包括主从同步延迟，Slave I/O延迟等。
- 2 自定义查询的时间范围及频率，默认监控30分钟内MySQL状态指标变化，可以根据自身需求扩大监控的时间范围，例如，监控1天内的状态指标变化。同时，监控的数据支持按照秒级、分钟级、小时级的粒度展示，支持在时间维度上对数据进行聚合、采样。
- 3 全面告警机制及渠道，自定义MySQL监控的告警策略及告警渠道，帮助第一时间发现MySQL的运行异常，包括自定义的告警策略和多样化的告警渠道，可通过平台告警、邮件告警、短信告警等。

MySQL监测实现原理

- 1 基于MySQL API获取数据，不入侵MySQL代码，只通过MySQL的API服务持续稳定地获取完整的状态信息。针对不同的监控需求，使用配置不同的API命令获取相关运行状态，同时解析和结构化运行状态信息，封装回传。原始数据由plugin解析，由plugin解析采样数据，同时记忆上次回传数据，以增量形式上报封装数据，节省了数据传输量，同时减少后端服务解析工作。
- 2 采用多层架构模式，通过API进行交互，耦合性较低，实现原理如图2-45所示。
 - 1) 数据采集，被监控的MySQL服务器上需创建一个专用的MySQL用户，plugin的数据采集模块通过此用户与MySQL建立连接，通过MySQL API命令获取MySQL运行时的状态，将数据回传给预处理模块。MySQL API命令包括：SHOW GLOBAL STATUS、SHOW SLAVE STATUS、SHOW MASTER STATUS、SHOW ENGINE INNODB MUTEX、SHOW ENGINE INNODB STATUS。
 - 2) 预处理，预处理模块对采集的数据进行初步加工，包括：筛选去除无关的监测指标，

统一化数据指标，计算当前状态值，如累加、均值、差值等简单数据操作，数据格式化，将众多数据指标规整化成统一格式，包括数据单位、数据值、数据标签等。

3) 数据上报，将格式化后的数据通过POST方式上报给数据处理平台。

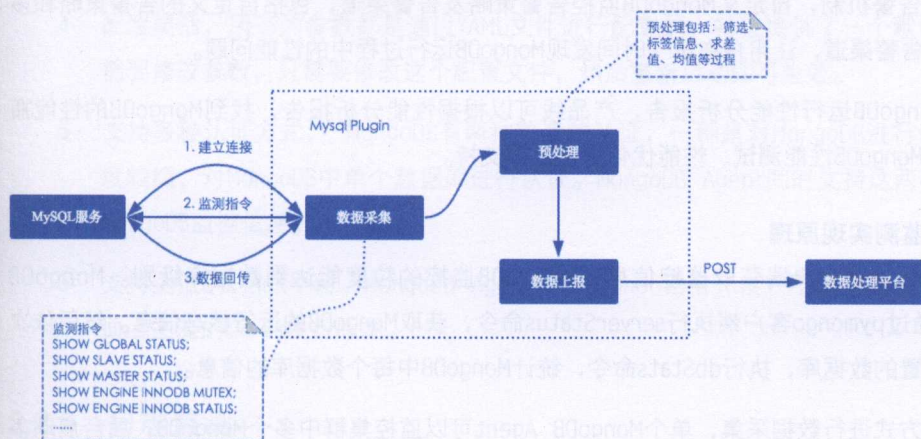


图2-45 MySQL监测实现原理

- 3 配置灵活，所有的参数都是通过Json文件进行配置的，且提供了一个默认值。如果用户需要监控多个MySQL，也仅需配置多个连接配置，然后重新启动plugin即刻生效。
- 4 支持主流的Linux平台，采用Java开发，性能稳定，损耗小。通过插件式监控，无须修改MySQL原有配置，合理封装采集模块和上报模块，耦合度低。

2.2.4.2.6 MongoDB监测

MongoDB是一个面向文档的数据库，每个文档都是schema-free的，相当灵活。2015年1月，DB-engines网站按照流行度对数据库进行了排名，MongoDB排名第五，获得了最佳数据库的荣誉。2016年1月，DB-engines网站再次排名时，MongoDB以306分的成绩进入第四名，超过PostgreSQL，成为流行度最高的非关系型数据库。NoSQL市场上，MongoDB市场份额也在逐年扩大，截止2015年12月，MongoDB占了将近60%的市场份额。另外，在国内的一些招聘网站上，很多开发工程师岗位都有“熟悉NoSQL（MongoDB、Redis等）优先”等要求。

MongoDB监测价值及目标

- 1 全面的MongoDB性能监测，MongoDB Agent通过MongoDB的Python客户端来获取运行状态指标值，获取的信息比较全面，包含数据库操作、底层文件操作、客户端、网络等各方面信息，另外还统计了MongoDB中各个数据库的信息。

- 2 实时监控MongoDB的各种运行状态指标,同时也支持对指标的历史信息进行查询。监控数据多粒度展示,监控的数据支持按照秒级、分钟级、小时级的粒度展示,支持在时间维度上对数据进行聚合、采样,并提供自定义的MongoDB监控频率。
- 3 完善的告警机制,自定义MongoDB监控告警策略及告警渠道,包括自定义的告警策略和多样化的告警渠道,让用户在第一时间发现MongoDB运行过程中的性能问题。
- 4 提供MongoDB运行性能分析报告,产品线可以根据性能分析报告,找到MongoDB的性能瓶颈,为MongoDB性能测试、性能优化提供数据支持。

MongoDB监测实现原理

- 1 基于Pymongo客户端获取监控信息,MongoDB监控的粒度能达到数据库级别。MongoDB Agent通过pymongo客户端执行serverStatus命令,获取MongoDB的运行状态信息。然后依次遍历配置的数据库,执行dbStats命令,统计MongoDB中每个数据库的信息。
- 2 多线程方式进行数据采集,单个MongoDB Agent可以监控集群中多个MongoDB,后台启动多个线程,每个线程采集一个MongoDB的数据,所有数据采集完毕后,统一通过POST方式上报。通过这种方式,减少MongoDB Agent的部署。
- 3 采用多层架构模式,通过API进行交互,耦合性较低,实现原理如2-46所示。

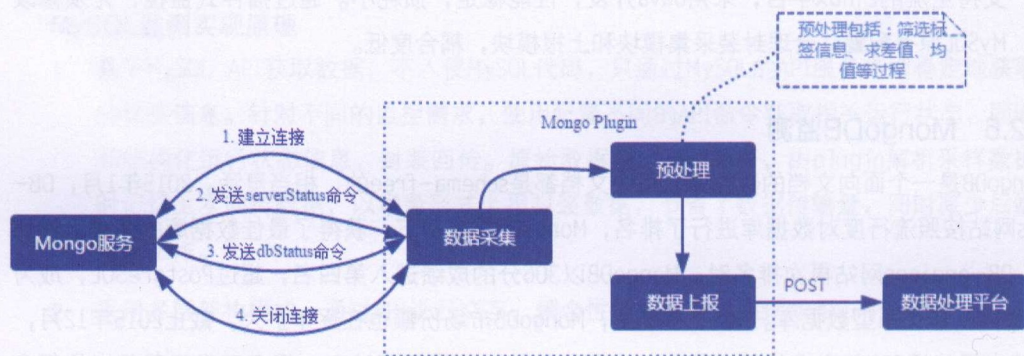


图2-46 MongoDB监测原理

- 1) 数据采集,数据采集模块通过MongoDB的Pymongo客户端来获取运行时的状态信息。Pymongo客户端首先与MongoDB服务建立连接,执行serverStatus命令,获取MongoDB服务的状态信息,然后遍历用户配置的数据库,针对每个数据库,执行dbStats命令,分别获取每个数据库的运行信息。
- 2) 预处理,预处理模块对采集的数据进行过滤,选择需要监测的指标。针对某些指标,

进行差值和均值运算。

3) 数据上报, 数据预处理完后, 数据上报模块给数据打上用户标识等信息, 然后通过POST方式上报数据处理平台。

4 配置灵活, 所有的参数都是通过YAML文件进行配置的, 并且提供了一个默认值。如果用户需要修改参数, 只需要修改这个配置文件, 然后重新启动即可生效。

5 支持多种认证方式, MongoDB有两种粒度的认证, 一种是对MongoDB进行认证, 另一种粒度较细, 对MongoDB中单个数据库进行认证。MongoDB Agent同时支持这两种认证方式获取MongoDB监控信息。

6 支持主流的Linux平台, MongoDB Agent采用Python语言开发, 不依赖系统相关库, 支持市面上主流的Linux发行版本。

2.2.5 日志监测

日志监测也是日志分析, 日志分析的能力决定了日志监测的效率和深度。在数据分析中, 有90%数据都是非结构化数据, 其中大部分的是日志。如用户、应用、运维、安全审计以及相关业务数据等都属于日志, 而目前对日志的价值挖掘成为大多数互联网公司的首要战略。企业需要建立统一化的数据分析, 提供最大化弹性的分析服务, 精准定位错误, 快速排查问题。

主流日志分析方案

1 Splunk。以网络安全、服务监控方面来看, 国外的Splunk提供了功能强大且纪录详细的日志分析。Splunk是基于原始日志数据(Raw data)内容建立索引, 保存索引的同时也保存原始日志内容, 以此可以快速定位错误日志。对于常规的日志分析也很强大、自由, Splunk对Apache、Squid、系统日志等进行索引, 然后可以交叉查询, 并支持复杂的查询语句, 最后通过直观的图表进行展示。值得一提的是, Splunk的日志搜索引擎非常强大, 从日志的产生到搜索分析出结果只有几秒钟的延时, 且它每天能够处理TB级的数据日志。同时, Splunk的分析查询也非常灵活, 采用准实时搜索的方法来分析日志, 功能十分丰富, 例如Search Processing Language, 它类似Linux命令, 支持管道, 子查询等功能。

2 Hadoop或NoSQL。就目前而言, 企业自身的日志处理是由企业自有的部门各自进行处理的, 较为常见的是, 每个业务部门都有自己的日志处理方式, 继而导致由业务的出发点不同而引发的各式各样的问题, 如数据结果不一致、数据管理混乱、数据迁移联动等问题。一些大公司为解决此类问题, 都会成立专门的大数据部门, 集中处理分析所有的日志及业

务数据。企业级的日志分析系统在发展过程中，经历了从日志1.0（利用数据库分析）到日志2.0（利用Hadoop或NoSQL分析）的阶段。而目前大部分的企业，基本都处于日志2.0的状态。

- 3 实时日志分析。对于大部分的初创公司，一般以上的架构已足以应付日增几十GB日志规模的离线数据分析需求。而对于有一定实时需求的公司，则还需要增加实时计算的框架。在开源领域，实时计算走在前列的框架主要是Storm、Spark Streaming（Spark系列）、Samza。在Storm中，主要先设计一个实时数据流向的拓扑图，称之为Topology。而Topology中主要角色分为Spout和Bolt，Spout负责发送tuple，Bolt负责处理tuple。而在Spark Streaming中，就不像Storm那样一次一个地处理数据流，而是处理前按时间间隔预先将其切分为一段一段的批处理作业，称之为DStream（DiscretizedStream），一个DStream是一个微批处理（micro-batching）的RDD（弹性分布式数据集）。而Samza处理数据流时，会分别按次处理每条收到的消息。Samza的流单位既不是元组，也不是Dstream，而是一条条消息。在Samza中，数据流被切分开来，每个部分都由一组只读消息的有序数列构成，而这些消息每条都有一个特定的ID（offset）。该系统还支持批处理，即逐次处理同一个数据流分区的多条消息。
- 4 OLAP。尽管数据平台化能解决大部分公司的数据报表等需求，但应对更深层次的价值挖掘依旧无力，纵使有Hive、Spark SQL、Impala等大数据分析工具，可对于数据分析人员来说长达分钟级的等待依旧是漫长的，且效率低下。目前初具规模的企业都会选择搭建OLAP（On-Line Analysis Processing）平台，这是一种共享多维信息的快速分析技术，能帮助分析人员快速、灵活地进行大数据复量的复杂查询，并且以一种直观、易懂的图表形式呈现给分析人员。开源社区里，目前关注度比较高的是eBay开源的Kylin OLAP分析引擎。Kylin是Hadoop平台中OLAP，于2014年10月正式开源，同年11月加入Apache孵化器，并于2015年11月正式毕业成为Apache顶级项目。

如何搭建日志分析平台

根据经历过的项目经验，日志分析架构如图2-47所示，主要分成服务监控层、数据计算层、数据存储层、数据集成层4层逻辑。每个逻辑层中包含若干流行开源服务组件，用来支撑海量日志计算、存储需求。日志分析模块及说明如表2-17所示。

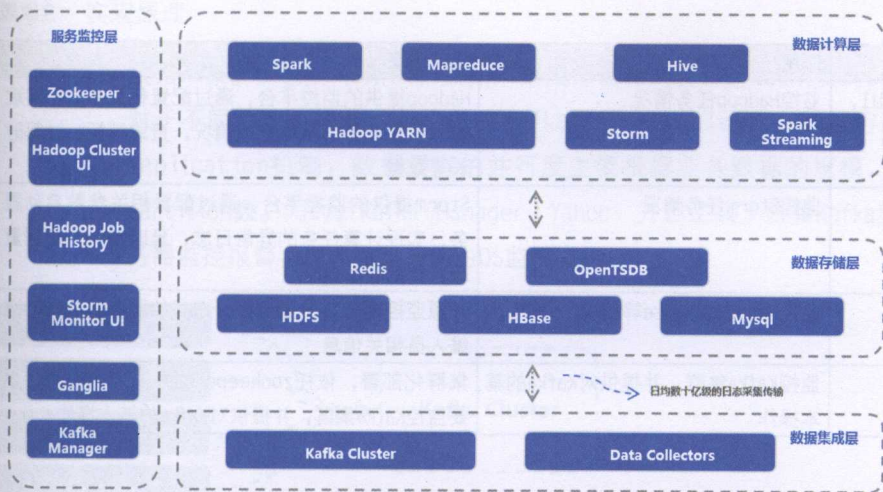


图2-47 计算、存储架构图

表2-17 日志分析模块及介绍

模块	作用	实现原理
Kafka cluster	缓冲 log_agent实时采集回来的数据	集群化部署Kafka，与zookeeper协作管理，实现日均数十亿级的数据缓冲
Data Collectors	收集汇总plugin监控数据及其他业务数据	基于OpenTSDB，实现时间序列的监控数据存储
HBase, OpenTSDB	存储各种监控数据及业务数据	开源框架，列存储式的NoSql
Redis	部分业务数据的缓存及秒级业务数据的主要存储设备	开源框架，NoSql
Mysql	Hadoop, Spark等离线计算的主要存储数据库	关系型数据库，主从库读写分离，主库写，从库读
HDFS	存储由Kafka2hdfs落地的原始数据，以及MapReduce的中间结果	Hadoop的分布式文件系统，为MapReduce和Spark等计算框架提供数据源
Hive	数据分析，应对客户的临时需求	基于MapReduce的类SQL分析框架，优化配置后快速应对客户的临时查询或分析需求
Hadoop YARN, Mapreduce	离线计算流量带宽，错误数，日志下载，日志分析等业务	集群化部署Hadoop以多个计算框架，通过定时任务调度系统主要负责长时间域的离线计算
Spark	离线计算5分钟级的流量带宽，错误数，访问量的业务	部署在Hadoop YARN上的计算框架，通过定时任务调度系统主要负责短时间域的，快速的离线计算
Storm, Spark streaming	实时计算流量带宽，错误数，单日汇总等业务，秒级监控服务器	集群化部署，Storm主要负责低延时的服务器监控，Spark streaming负责大数据量的流量带宽等实时计算业务
Zookeeper	协作Hadoop, storm, Hbase等分布式开源框架	集群化部署，协作Hadoop, Storm, Hbase等分布式开源框架

续表

模块	作用	实现原理
Hadoop cluster UI, Hadoop job history	监控Hadoop任务情况	Hadoop提供的监控平台，通过配置相关参数，实现提供计算任务的具体完成情况，资源消耗，计算时长等数据
Storm Monitor UI	监控Storm任务情况	Storm提供的监控平台，通过配置相关参数启动服务，实现计算任务的异常日志，延时情况，数据量级，负载等数据
Ganglia	监控Hadoop，HBase等情况	开源监控框架，主要监控分布式的服务，提供给运维人员相关信息
Kafka Manager	监控Kafka集群，并提供对Kafka的基本操作	集群化部署，依托zookeeper的开源监控框架，主要监控Kafka集群，并提供对Kafka的基本操作

1 架构说明

- 1) 数据集成层，数据集成层主要汇集集成数据源，供各种组件使用（例如数据获取、数据整理等）。目前，有kafka2hdfs、kafka2opentsdb服务分别落地新增的数据到HDFS、OpenTSDB，以多线程模式并行处理。Collector主要设计为数据收集、数据适配、数据分发，将上传的plugin数据收集后适配成OpenTSDB所需的数据格式，然后数据分发到TSD进行数据落地。
- 2) 数据计算层，数据分析层由实时计算、离线计算、OLAP分析组成。实时计算目前由Storm搭建，计算结果pipeline写入Redis，过期时效为数分钟。离线计算目前由MapReduce计算框架负责，Job调度由基于Quartz开发的JobScheduler定时调度。OLAP分析基于长期建立的数据仓库，对每日新增数据进行预计算，更新维度索引，提供弹性的数据分析。OLAP则基于Kylin分析引擎进行研发，提供构建cube，查询cube，数据适配，数据同步等服务。
- 3) 数据存储层，数据分析存储层存储数据分析结果或者中间结果，由后续数据服务提供简单聚合等计算。Redis负责实时数据的结果存储（过期失效），以及调度状态、任务成功失败标记等值。Mysql主要负责时效性较长、数据量不大的计算结果，之后会对冷热数据进行分离，对历史数据存入HBase，较新的数据存入Mysql。Hive和HBase主要负责时效性长、数据量大的计算结果，之后会存入各种预计算的结果、中间表、长期保存的数据，包括监控数据、报表数据等。
- 4) 数据服务层，数据服务层主要负责应用层的服务请求，采用微服务的架构体系，Docker部署，服务不相互依赖或简单依赖，提供各种监控数据、报表服务。

2 实现原理

- 1) 数据采集。如图2-48所示,数据采集由部署在每台log_agent实时push到Kafka集群。对于不同的业务采集,相应的在Kafka集群中建立Topic,数据容灾主要依托Kafka replication机制,数据缓存的并行度主要根据业务数据的规模,创建与之匹配的partition数。同时,Kafka Manager (Yahoo! 开源工具) 协管Kafka集群,而集群的服务器监控报警,进程监控报警由OS监控提供。

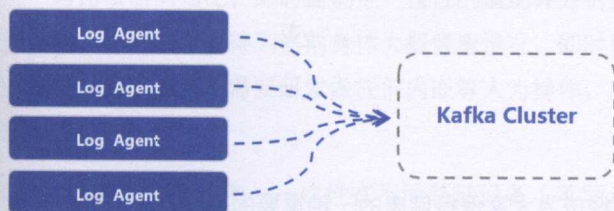


图2-48 数据采集流程

- 2) 离线、实时数据分流。如图2-49所示, Kafka集群实现数据缓冲, Kafka2hdfs负责及时消费落地数据, Storm集群负责实时消费数据, 由此实现流分离。

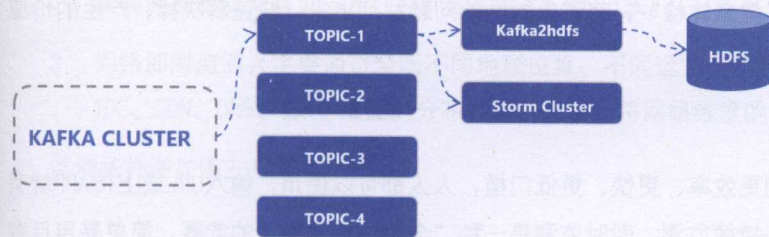


图2-49 离线、实时数据分流

- 3) 实时数据处理。如图2-50所示, Storm集群负责实时展现业务, 针对不同的业务规模和业务需求使用不同的API。

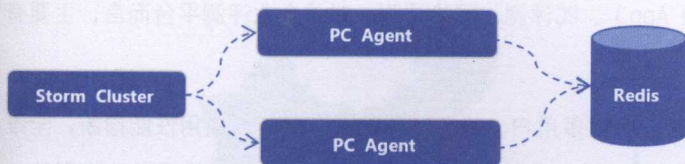


图2-50 实时数据处理

- 4) 离线数据处理。如图2-51所示, 离线处理, 主要集中对落地后的数据进行数据清洗, 预处理, 统计, 分发等, 业务涵盖常规的流量带宽计算, 日志下载, 日志分析, 质量统计, 监控数据服务等, 对计算处理后的数据存入到Mysql、HBase (hdfs)、Redis等。

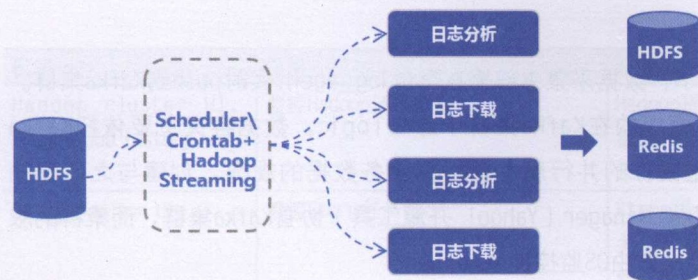


图2-51 离线数据处理

2.3 应用性能即时监测

应用性能即时监测与体检类似，体检已经成为大家保持健康的一种重要的手段，人们对自己健康的管理，重点应该是对疾病的早期发现，而疾病的早期发现主要还是依靠体检。对健康重视的人，都会定期体检，并且重视体检报告，根据各科体检结果，纠正不良生活习惯，达到预防和改善健康的目的。应用性能即时监测达到的目的与日常体检具有相同的意义，每一次对产品进行一次评测如何对产品做一个全面的“健康体检”，将表象的性能问题罗列出来，快速解决掉，产生的价值是巨大的，并且与持续监测互补。

即时监测的基本特性

相比持续监测，即时监测更高效、更快、更低门槛，人人都可以使用，输入URL或上传APP就有结果，但深度和可持续性弱于持续监测。即时监测是一种“全民性能管理”的武器，简单易用且容易普及，无论产品、研发、测试还是运维都可以在产品迭代或上线时进行性能分析，通常即时监测具备多视图、全方位分析产品在不同地理位置、网络、设备、浏览器、分辨率下的性能瓶颈，并自动提供行之有效的优化建议，主要覆盖前端、后端、系统、网络、安全、速度等维度性能，按类别主要分为移动评测（Web App、Native App）、PC评测、网络评测。对于企业评测平台而言，主要有以下特性：

- 1 提供跨终端全维度的应用诊断，一键多用户、终端、浏览器、网络、应用性能诊断，全维度优化建议，加快问题诊断速度，自动化执行基准并预测异常，发布前规避性能影响。（评测发生在事先准备好的一批真实的PC、移动设备上）
- 2 自动化快速评测加快产品推向市场的速度，增强敏捷性、提高应用的质量、减少瓶颈并削减运营成本。

- 3 全维度的数据分析，深度分析并快速识别所有应用性能领域中的瓶颈问题，大幅提高研发效率。
- 4 开放API可以融入到各产品团队研发流程及公有云、公共平台，最低门槛实现应用性能评测能力。

即时监测分类

与持续监测相比，即时监测是一次性的监测并分析监测对象的问题和瓶颈，即时监测好比去医院体验，体验后可以得到当前身体大概健康情况，即时监测与持续监测最大的不同是，即时监测一次性出结果，而且不需要研发做任何内嵌等人为操作，属于基于真实用户的主动监测，主要分为以下3类：

- 1 移动即时监测，一次性在不同移动设备（不同iOS和Android操作系统和不同厂商不同款手机和平板电脑、不同运营商、不同屏幕大小等）上监测Web App和Native App加载性能，并将存在的性能问题分析出来。
- 2 PC即时监测，主要通过不同屏幕及不同浏览器厂商、不同版本及不同运营商等环境下进行真实浏览器渲染网页应用，将问题和瓶颈分析出来。
- 3 网络即时监测，主要通过全国不同地理位置、不同运营商的真实用户，分析应用在全国的IDC、CDN、DNS、ISP等维度的分布和解析策略，将网络维度的问题体现出来。

评测拓扑图如图2-52所示。

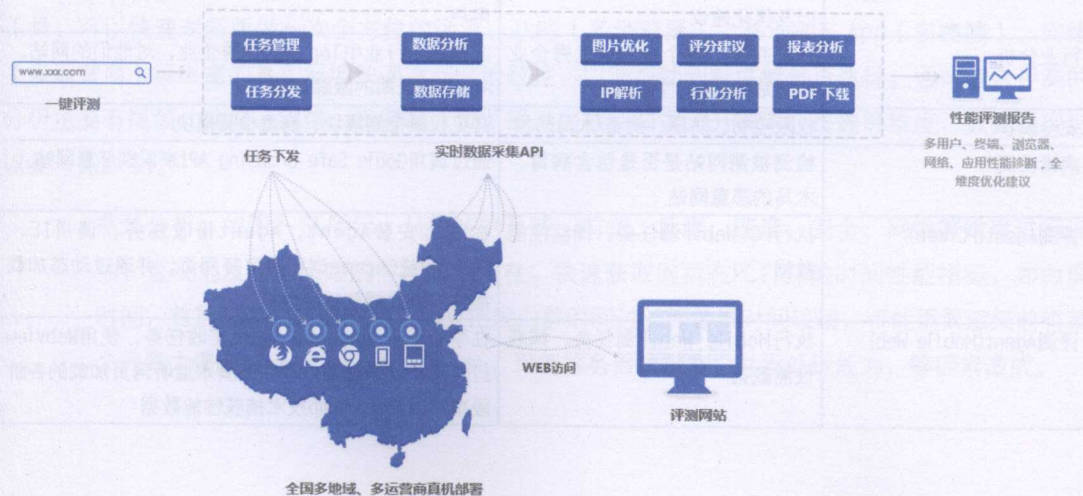


图2-52 即时监测拓扑图

评测架构及实现原理如图2-53和表2-18所示。

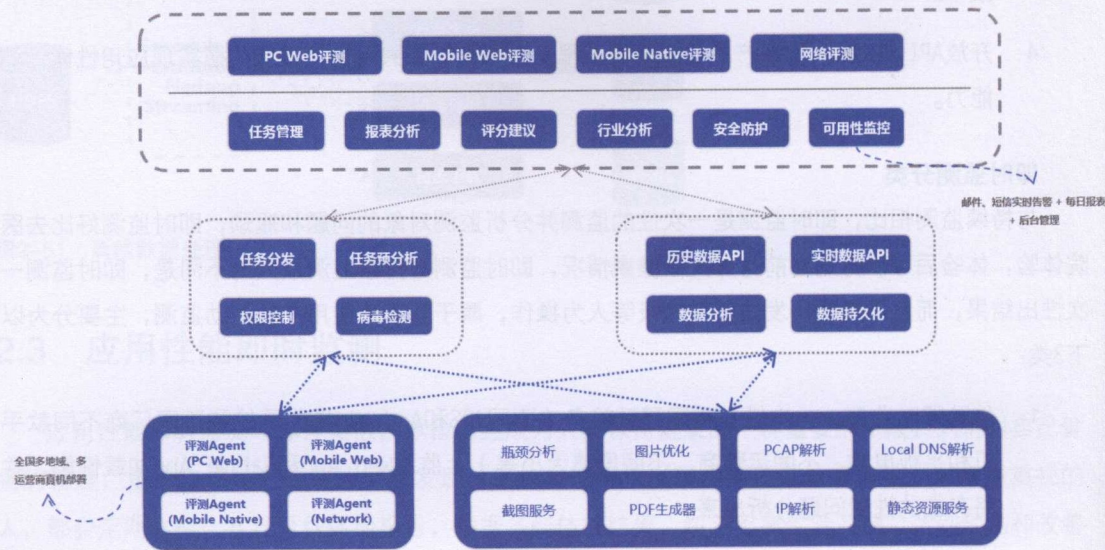


图2-53 即时监测架构图

表2-18 即时监测模块及说明

模块	作用	实现原理
报表分析	展示评测结果中的各项性能数据	将评测所得的性能数据通过前端系统展现给用户，使用到echart等图表类库
评分建议	展示被测网站、App的各项评分以及优化建议	将数据分析模块所得的评分建议通过前端系统展现给用户
行业分析	分析并展示各个行业中优秀企业的网站和App的性能情况	选择各个行业中Top N的优秀企业，对他们的网站、App进行长期的跟踪评测
安全防护	权限控制、防刷、病毒/木马防护	详见权限控制模块、病毒检测模块
病毒检测	检测被测网站是否是包含病毒、木马的恶意网站	通过调用Google Safe Browsing API来识别恶意网站
评测Agent(PC Web)	执行PC Web评测任务，捕获性能数据	在PC上安装Agent，Agent接收任务，调用IE、Firefox或Chrome浏览器打开网页，并通过动态加载插件的方式捕获性能数据
评测Agent(Mobile Web)	执行Mobile Web评测任务，捕获性能数据	在手机上安装Agent，Agent接收任务，使用WebView打开网页，通过插入评测JS脚本监听网页加载的各阶段事件以及tcpdump技术捕获性能数据

续表

模块	作用	实现原理
评测Agent(Mobile Native)	执行Mobile Native评测任务, 捕获性能数据	在手机上安装Agent, Agent接收任务, 下载、安装并运行被测App, 并同时采集App执行过程中的CPU、内存、流量、帧率等性能数据。通过代理技术捕获HTTP请求信息, 通过代理伪造证书技术捕获HTTPS请求信息
评测Agent(Network)	执行网络评测任务, 捕获IDC、CDN等网络分布与解析数据	在PC上安装Agent, 执行网络评测任务, 通过Ping、Traceroot、Dig等命令, 获取域名解析情况
图片优化	在不影响视觉效果的情况下对图片进行压缩优化, 分析被测网站图片资源是否优化得当	针对不同的图片格式, 调用不同的优化工具进行优化
PCAP解析	解析pcap文件获取HTTP请求信息, 生成HAR格式的文件	对开源的PCAP Python解析项目进行二次开发
Local DNS解析	获取域名在不同地域的解析情况	指定Local DNS对域名进行Dig
PDF生成器	将给定的网页转化为PDF文档	通过调用开源的wkhtmltopdf完成转换
IP解析	对给定的IP进行解析, 获取国家、省份、城市、运营商等信息	通过对IP库进行查询
静态资源服务	提供静态资源的存储和在线访问功能	通过FastDFS集群提供服务
实时数据推送	提供评测时的实时数据推送服务	通过SockJS进行实时数据推送服务, 兼容各个版本浏览器, 智能选择WebSocket、Flash或轮询技术

即时监测视图

即时监测最大的优点是一键出结果, 通常产品线每天都有多次新版本发布, 很需要一种评测工具, 可以快速对新版做一次全方位的评测, 从PC (多浏览器)、移动Web App (多终端)、网络 (多运营商) 等丰富的真实环境快速得到性能数据, 以便帮助判断性能是否达标。通常综合评测的分析主要有瓶颈、速度、网络、运营商、浏览器、内容、错误、分辨率、终端等维度, 数据展现可以参考图2-54。

- 1 瓶颈分析。如图2-55所示, 从网站的系统、前端、后端、速度、安全、网络等维度进行评估, 直观地反映出网站的性能瓶颈所在。快速获取网页在PC打开的时间性能指标, 如白屏时间, 首屏时间, 后端时间。如果出现白屏时间过长而首屏时间较短, 可能页面渲染前有某个元素太慢造成卡顿。后端时间过长, 可能服务器端带宽, 并发处理能力, 等因素造成。

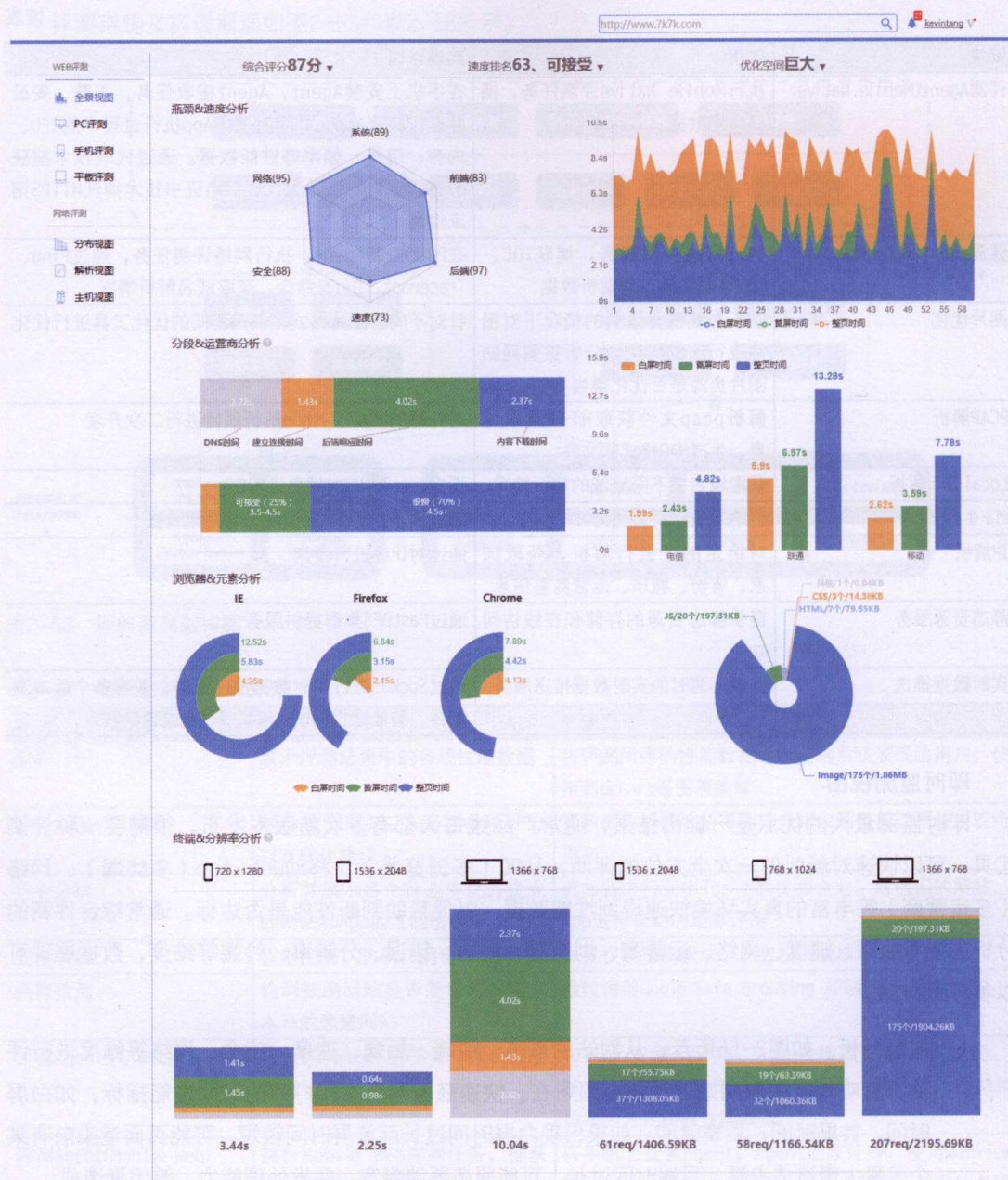


图2-54 综合视图

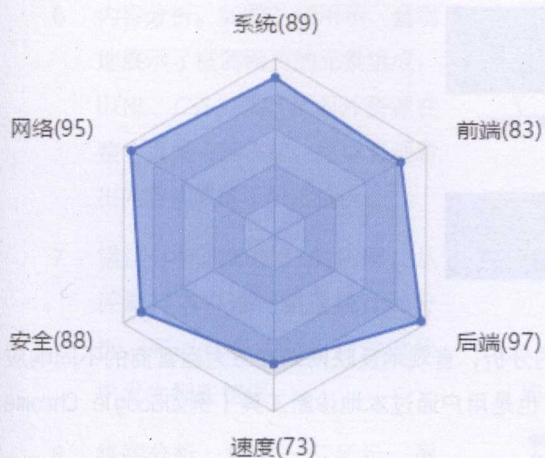


图2-55 瓶颈分析视图

2 速度分析。如图2-56所示，将评测所有样本的打散成趋势视图展示，从中可以看出在该并发下，网站的关键性能指标（白屏时间、首屏时间、整页时间）的性能情况。

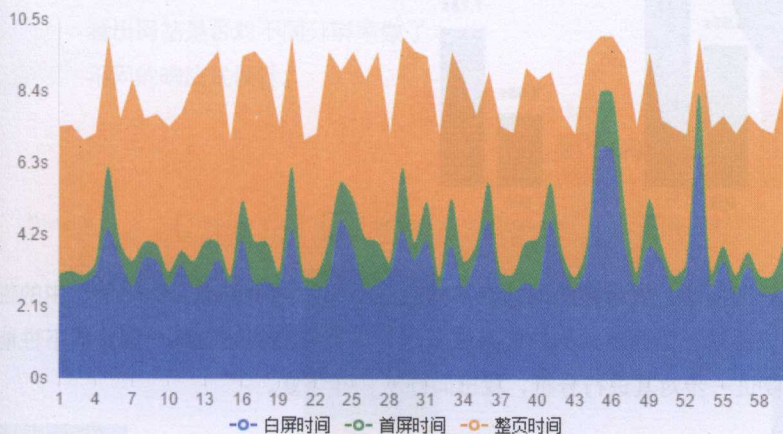


图2-56 速度视图

3 网络分析。如图2-57所示，将真实的生产环境网络和用户解析可视化。首先将所有页面请求的过程分为四段（DNS时间、建立连接时间、后端响应时间、内容下载时间）进行统计，诊断网站的性能瓶颈所在。同时，将所有评测结果进行统计，可以直观看出快速和慢速比例等情况。

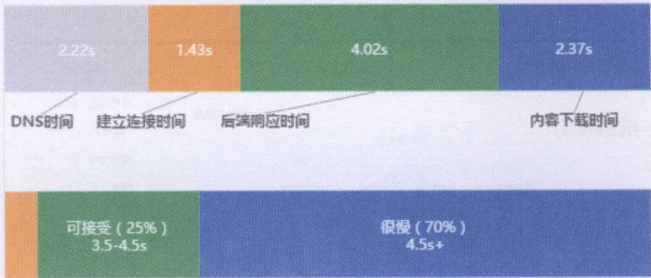


图2-57 网络指标视图

4 运营商分析。如图2-58所示，对运营商的分析，直观地反映网站在三大运营商的不同响应速度，分析服务器运营商分布是否合理，也是用户通过本地诊断工具（例如Google Chrome Dev Tools或Fire Bug）所无法实现的功能。

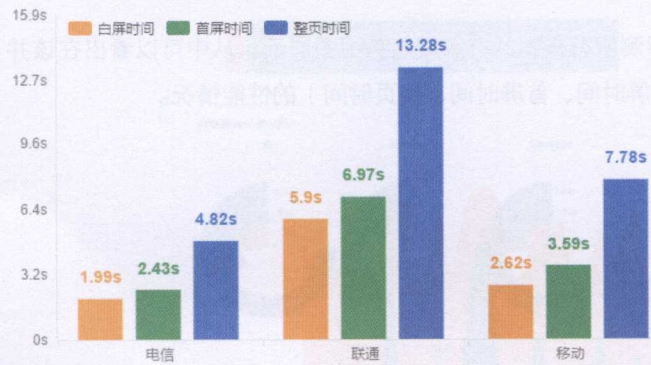


图2-58 运营商视图

5 浏览器分析。如图2-59所示，展示被测网站在不同浏览器（IE、Firefox、Chrome）中的性能差异。不同的浏览器对于页面渲染的性能存在差异，如果被测网站在某一浏览器下性能有显著下降，则需要进一步对其进行分析，找出问题所在并修复。

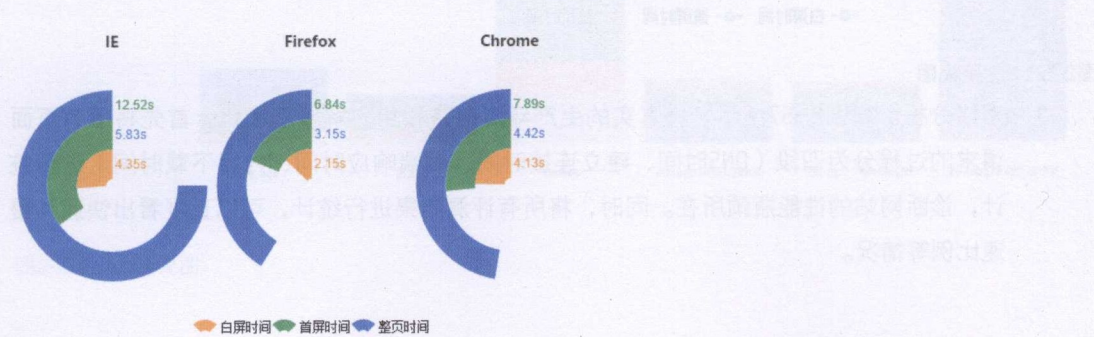


图2-59 浏览器视图

6 内容分析。如图2-60所示，直观地展示了被测网页的元素组成，HTML、CSS、JS以及图片资源在整个页面中的占比，可以直观看出内容构成的不合理情况。

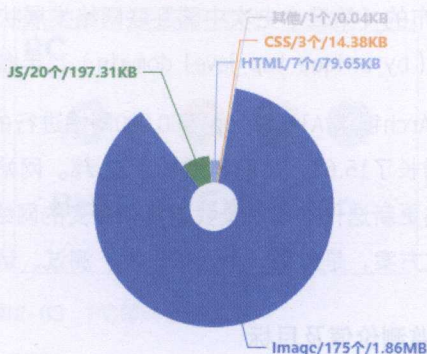


图2-60 内容视图

7 错误分析。如图2-61所示，从评测样本中进行错误统计和分析，可以直观地看出评测过程中发生那些错误。

8 终端分析。如图2-62所示，展示了被测网站在不同分辨率终端下的响应速度及内容构成，可以看出在手机、平板、PC主流分辨率上的速度差异，也可以直观地看出网站是否对不同分辨率做了不同的响应式设计。

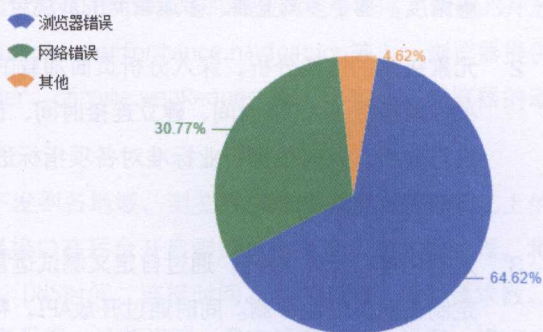


图2-61 错误视图

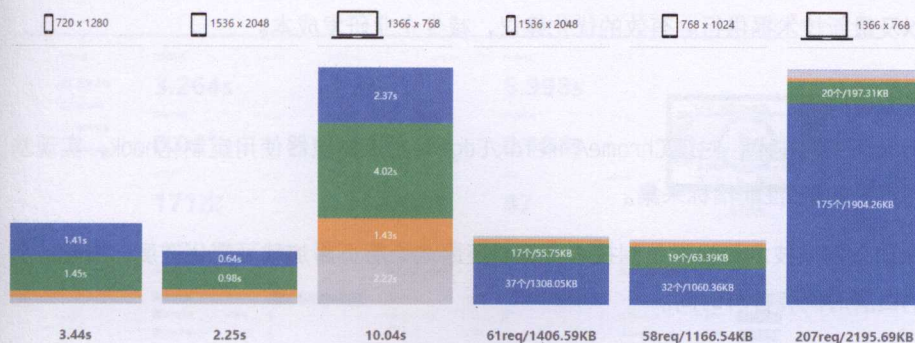


图2-62 终端视图

2.3.1 PC即时监测

据Netcraft发布的《December 2015 Web Server Survey》，截至2015年12月，全球共有9.01亿个在线网站（by unique hostname），其中活跃的网站数为1.72亿。据中国互联网络信息中心

(CNNIC)发布的《第三十七次中国互联网络发展状况统计报告》，截至2015年12月，中国网站总数为423万个 (by unique top level domain)，年增长26.3%，网页总数为2123亿，年增长11.8%。

据HTTP Archive对Alexa Top 500,000网站进行的统计，截至2016年1月1日，全球网页平均大小比去年同期增长了15.6%，比前年增长了32.7%。网站的内容和功能日益丰富，再加上激烈的市场竞争导致的网站更新迭代加速以及我国复杂多变的网络环境，如何快速了解网站的性能情况并采取行之有效的优化方案，是对每一个网站研发、测试、运维及产品经理等IT从业人员的考验。

PC即时监测价值及目标

- 1 帮助产品线提升产品质量。只需要输入一个URL，从系统、网络、前端、后端、速度、安全等维度，基于多浏览器、多运营商的环境全方位评估网站性能，让网站性能问题无所遁形。
- 2 元素级别的性能分析，深入分析页面加载的每一个细节，对每一个网页元素的正确性和各项性能指标（如DNS时间、建立连接时间、首包时间、下载时间、gzip压缩、图片压缩等）进行分析，同时根据行业标准对各项指标进行评分，并根据业界最佳实践对性能问题提出行之有效的解决方案。
- 3 定制化接口及开放API，通过自定义测试运营商、地域、浏览器、分辨率等，帮助企业实现定制化网页性能评测。同时通过开放API，帮助企业快速实现网页评测的自动化，增强敏捷性，在保证产品质量的前提下，加快产品推向市场的速度。
- 4 让性能优化变得简单、高效。专业的大数据分析，快速定位网站性能瓶颈，并根据业界最佳实践以及最新技术提供行之有效的优化建议，减少企业研发成本。

PC即时监测实现原理

- 1 基于动态hook注入技术，对IE/Chrome/Firefox/Edge等主流浏览器使用定制化hook，实现基于真实浏览器的Web性能指标采集。
- 2 基于图像动态扫描技术和视屏录制技术，完整还原渲染层页面加载可视化进度，准确的反应页面的白屏时间和首屏时间。
- 3 自定义脚本支持，支持在评测的过程中执行用户自定义的脚本，可实现自定义测试流程，比如先执行登录脚本，再进行测试。
 - 1) 评测客户端通过调用用户本地安装的真实浏览器，模拟用户访问页面，从而获取最真实的性能数据。
 - 2) 评测客户端可方便地部署在任意一台PC上，从而可以进行跨地域、跨运营商、跨浏览

器、跨分辨率等定制化测试。

4 实现原理如图2-63所示。

- 1) IE浏览器,通过BHO技术(微软推出的作为浏览器对第三方程序员开放交互接口的业界标准),动态控制浏览器的行为(如打开指定的网站等),同时安装钩子监控网页访问的过程以获取性能数据。



图2-63 PC即时监测原理

- 2) Chrome、Firefox等其他浏览器,通过JavaScript编写的浏览器扩展程序,调用JS开放的Performance API (performance.timing、performance.navigation等)、浏览器提供的交互与调试API (如chrome.debugger、chrome.webRequest等),实现与浏览器的动态交互以及获取页面访问的性能数据。

在即时监测过程中,网站URL会从服务端下发到各地域、浏览器、网络运营商的真实PC上的监测客户端,客户端会调用Windows原生的浏览器接口在后台开启浏览器完成真实的加载过程。将核心性能指标如白屏时间、首屏时间、整页时间、DNS时间、连接时间、后端时间、页面请求数、页面总大小等传回服务端进行可视化。并通过元素视图、诊断视图、瀑布视图、评分视图、视频回放等进行深度分析,如图2-64所示。



图2-64 PC评测视图

2.3.2 移动Web App即时评测

移动流量正在迅速增长，对众多消费者而言，手机和平板已经成为他们浏览网页的主要入口，但其性能表现却差强人意。Compuware公司委托Equation研究所做的一项研究表明，46%的移动用户表示他们手机上的网站加载速度过慢。由Strangeloop发起的一项针对200家领先的电商网站研究表明，3G网络下平均加载时间为11.8s，4G环境下只有轻微的改善，为8.5s。

移动网络的特殊性（网速慢、延迟大、不稳定）和移动设备的多样性（不同分辨率、不同操作系统、不同浏览器、不同硬件配置），给移动网站的开发、测试和运维等工作带来了极大的难度和不便。移动网站的性能问题比PC更加严重，需要更多的重视和关注。移动Web技术的发展异常迅速，使用移动技术，可以开发出媲美Native App的轻应用和游戏。但这也使得移动网站的体积越来越大，功能越来越复杂，暴露出移动网站更多的性能问题。

移动Web App即时监测价值及目标

- 1 为移动网站量身定制的性能评测工具，帮助企业解决在新建移动网站或从PC网站向移动网站迁移过程中出现的各项性能问题，减少研发、测试成本。
- 2 让性能优化变得简单、高效，丰富的视图分析，快速定位移动网站性能瓶颈，并根据业界最佳实践以及最新技术提供行之有效的优化建议。与行业内的龙头企业以及行业标准作对比，了解自己与竞争对手在移动网站的用户体验上的差距。
- 3 针对移动网站的特性，针对移动网站特殊的网络环境以及移动网站特有的性能指标进行评测，尤其是HTML 5、Single Page等特殊Web应用，最大限度地暴露移动网站的性能问题。
- 4 定制化接口及开放API，同时通过开放API，帮助企业快速实现网页评测的自动化，增强敏捷性，在保证产品质量的前提下，加快产品推向市场的速度。

移动Web App即时监测实现原理

- 1 基于主流手机、平板设备及操作系统的移动Web App、移动API性能评测。从数十个最新的移动性能维度进行全方位评估，最大可能诊断瓶颈，给出优化建议。丰富视图还原真实原生态操作系统及浏览器渲染过程，问题一目了然。
- 2 前沿的网络探测，通过在远程手机客户端上执行ping、tracert、dig、nslookup等网络命令捕获网站的网络分布和性能，通过tcpdump、代理服务器、WebView事件接口以及JS注入等技术，获取请求的各项性能指标。
- 3 定制化简单，自主研发的移动Web评测客户端可配置性强，可指定评测的运营商、地域、

设备、分辨率、网络类型、并发数等，方便为企业量身定制评测方案。

- 4 高覆盖率，移动Web评测客户端支持主流的操作系统版本，主流的手机和平板设备，同时采集的数据指标非常丰富，包含移动Web特有的性能指标，性能问题一网打尽。

1) 评测客户端通过调用手机上安装的真实浏览器，模拟用户访问页面，从而获取最真实的性能数据。

2) 评测客户端可方便地部署在任意一台手机上，从而可以进行跨地域、跨运营商、跨设备、跨分辨率等定制化测试。

- 5 实现原理如图2-65所示，通过WebView的事件API以及JavaScript的动态注入技术获取浏览器的交互与性能数据，同时，通过tcpdump抓包技术以及对SSL底层API（SSLRead & SSLWrite）的监听追踪被测网页的HTTP/HTTPS请求详情。

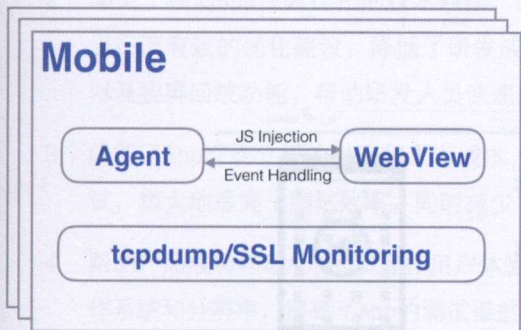


图2-65 移动Web App监测原理

移动Web App主要通过手机和平板设备加载，可以理解为小版的PC网页，因不同的硬件、分辨率和浏览器渲染方式等原因，导致无论是否适配，相比PC内容加载是完全不一样的。在即时监测过程中，移动Web App会从服务端下发到各地域、厂商、制式的真实手机、平板设备上的监测客户端，客户端会调用原生操作系统的接口在后台开启浏览器完成真实的加载过程，将核心性能指标如白屏时间、首屏时间、整页时间、DNS时间、连接时间、后端时间、请求数、页面大小等返回服务端做可视化。并通过元素视图、诊断视图、瀑布视图、评分视图、视频回放等进行深度分析，如图2-66和图2-67所示。



图2-66 手机评测视图



图2-67 平板评测视图

2.3.3 移动Native App即时评测

据全球领先的互联网统计公司comScore最近发布的报告，美国移动终端和PC端的使用占比分别为61%和39%。在2015年3月，仅使用手机上网的用户超过了仅使用PC上网的用户。对于排名前100的

数字媒体，其仅使用手机的用户比例高达49%。2015年6月，全美用户使用手机浏览器浏览网页的时间为118299分钟，比2013年同期增长了53%，而使用PC的时间比2013年同期仅增长了16%。

据全球领先的互联网统计公司comScore最近发布的“The 2015 U.S. Mobile App Report”，2015年6月，针对数字媒体，全美用户的移动App使用时间为778954分钟，比2013年同比增长90%以上，是移动浏览器使用时间的6.58倍，是PC端使用时间的1.41倍。

移动Native App即时监测价值及目标

- 1 降低了移动App性能监测门槛，测试、运维等人员无须专业的Android或iOS技术背景即可测出App的兼容性问题 and 性能问题。通过HTTP请求探测，不仅仅对CPU、内存、流量进行探测，同时对App运行过程中发出的HTTP请求进行跟踪探测，通过对请求的响应时间以及请求各阶段时间消耗的分析，定位由网络请求延迟产生的性能问题。
- 2 降低了移动App性能优化的技术门槛，基于丰富的视图，对测试的性能数据进行分析，提出行之有效的优化建议，降低了研发成本。快速问题追踪，通过查看log、Exception分析以及视屏回放功能，帮助研发人员快速追踪App运行效果、定位兼容性和性能问题。
- 3 降低了App性能分析的时间和人力成本，加速App的发布更新。多台设备同时进行自动化测试，极大地提高了测试效率，同时减少了人力成本。
- 4 帮助产品线将移动App的质量和用户体验做到极致。基于主流手机和平板设备，覆盖多个操作系统和分辨率，提高了App的测试覆盖率，最大限度发现和解决APP的兼容性和性能问题。

移动Native App即时监测实现原理

- 1 强大的网络分析，不仅仅支持对流量消耗以及HTTP请求的探测，还可以对App的后端Web服务的网络分布和质量进行分析。
- 2 兼容HTTPS请求分析，应对当前全站HTTPS的发展趋势，在支持HTTP请求分析的同时，通过对底层SSLRead和SSLWrite方法的监听，实现了对HTTPS请求的探测和分析。
 - 1) 同时支持移动Native App以及Hybrid App的兼容性和性能评测。
 - 2) 安装评测终端的手机和平板设备无须通过USB与电脑相连接，评测客户端可方便的安装在任意一台Android设备上。
- 3 实现原理如图2-68所示，对被测App进行随机Monkey测试，或运行定制的Robotium测试脚本进行测试。在安装、运行、卸载App的过程中：

- 1) 通过tcpdump抓包以及对SSL底层API (SSLRead & SSLWrite) 的监听实现对HTTP/HTTPS 请求的追踪。
- 2) 通过调用Android API (如ActivityManager. getProcessMemoryInfo等)、运行Linux命令 (如dumpsys等)、读取Linux进程信息文件等方法获取到被测APP的CPU利用率、内存占用、流量消耗等数据。
- 3) 通过screencap、screenrecord等工具对测试过程进行截图和视频录制。
- 4) 通过对测试过程中产生的log对APP崩溃、代码异常等情况进行分析。

在即时监测过程中, 移动Native App会从服务端下发到各地域、厂商、制式的真实手机、平板设备上的监测客户端, 客户端会调用原生操作系统的接口安装Native App、启动Native App, 并模拟用户进行一系列的点击操作, 将核心性能指标如安装时间、启动时间、帧率、CPU 占用率、内存占用、流量消耗、请求大小、首包时间、响应时间等传回服务端进行可视化。并通过性能分析、网络分析、视频回放等进行深度分析, 如图2-69所示。

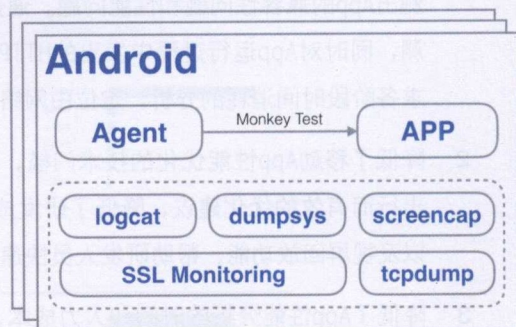


图2-68 移动Native App监测原理

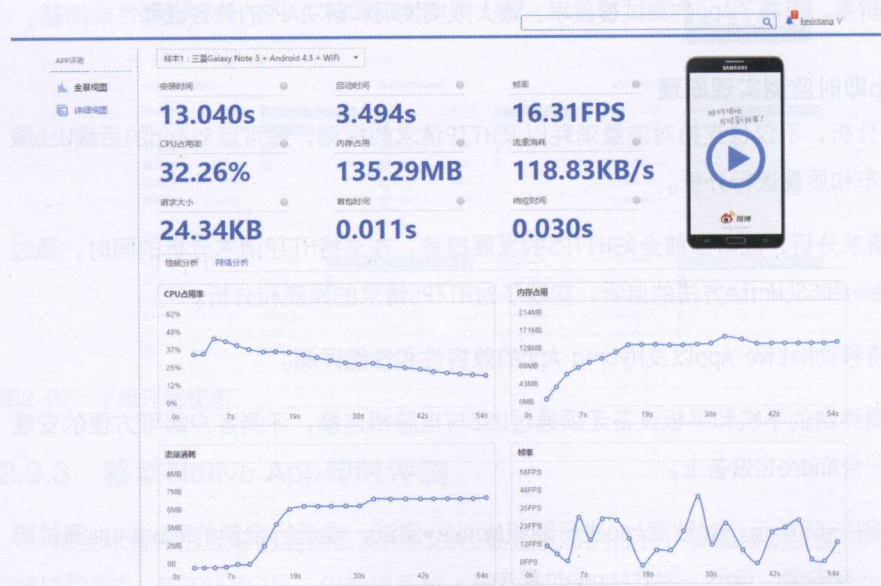


图2-69 性能分析视图

2.3.4 网络即时监测

中国网络情况复杂，基本南北格局一直困扰整个互联网。每个运营商都优先保障本网内访问速度，而跨运营商的访问体验，并不是他们的业务重点，这种利害关系就导致跨运营商问题，成为一个难以解决的客观问题。根据CNNIC发布的《域名根服务系统运行态势与分析报告》，共计探测根域名服务2,235,480次，其中返回结果为优者占28.06%，返回结果为良者占37.87%，返回结果为差者占34.07%，中国部分地区访问根服务仍然未能命中中国镜像，不同省份运营商访问质量不均衡。所以，Web应用几乎无法避免跨运营商解析的问题，用户或者开发者有实时了解全网解析的需求。

另一方面，国内CDN市场需求巨大，但厂商技术、标准参差不齐，信息不对称，CDN使用方急需第三方评测/监测数据支撑。随着中国的互联网产业迅速发展，中国对于CDN的需求总量在不断增加，基于不同客户的定制化CDN服务也将成为发展的趋势，新互联网应用的增加将带来更加复杂的CDN加速需求，因此客户一定会使用最稳定、效率最高、价格最便宜的那个作为主要的CDN提供商。因此，作为CDN使用方（终端用户或者CDN代理服务商），都需要第三方的网络解析数据，借此判断CDN服务的优劣。

网络即时监测价值及目标

- 1 基于亿级别的各产品线真实用户数据得到运营商、省份、用户的分布属性信息，并智能分析IDC、CDN、ISP、DNS、TDO等网络分布、调度解析合理性。实时分析CDN厂商、自建CDN及全国IDC节点性能，秒级监测每一个节点延时和可用性。
- 2 帮助产品线快速了解自己产品或者竞品网站的网络拓扑结构，快速分析使用的CDN厂商以及各IDC节点分布（具体至服务器IP）以及CDN节点质量。
- 3 智能分析IDC、CDN、ISP、DNS、TDO等网络分布，并进行评分。动态分析网站使用的CDN厂商及其分布，直观了解CDN质量。真实网络环境和用户解析可视化地图展现，快速判断网络跨运营商、跨地域访问、访问过慢等调度、访问异常情况。
- 4 通过积累的行业数据，以及网页爬虫技术，算法过滤掉网页中分享、广告链接，并且实时分析出评测网站使用的CDN厂商。数据进行投票机制，支持不同渠道采集数据按权重投票，渠道/权重可灵活配置，最终选出最真实可信的解析结果。
- 5 监测端接入方便，采用标准MQTT协议，方便接入不同渠道的客户端，并支持按照地域、运营商按需下发网络评测任务。通过全国数万招募客户端以及CDN节点对网站进行网络分析，做到地域、运营商全覆盖。

网络即时监测实现原理

- 1 Agent订阅，在全国服务器及IDC、CDN节点上部署Agent，Agent通过实现MQTT订阅/发布协议的mosquitto lib在服务端进行订阅，接收服务端的任务。
- 2 任务下发，服务端接收到用户的网络评测请求之后，首先会对请求的URL进行检查，其中包括：URL可访问性、页面元素统计、CDN链接判断。然后，一方面对于所有订阅的Agent，通过MQTT协议下发该URL以及页面对应的监测URL，使用ping和dig命令进行网络解析；另一方面，请求全国不同区域的LocalDNS服务器，解析DNS服务器返回的结果。
- 3 数据上报，Agent接收任务后，通过HTTP的方式上报数据至服务端。
- 4 数据接收，服务端接收到数据之后，将数据根据请求ID传入对应消息队列；服务端下发任务之后，就一直监听消息队列，获取网络解析样本。
- 5 数据统计，服务端网络解析数据由两部分组成，分别是不同地区的LocalDNS和部署在CDN节点的Agent。当服务端接收到足够多的网络解析样本或者超过采集时间，就开始对样本进行统计，统计完成之后再将数据通过前端进行展现。

网络即时监测视图

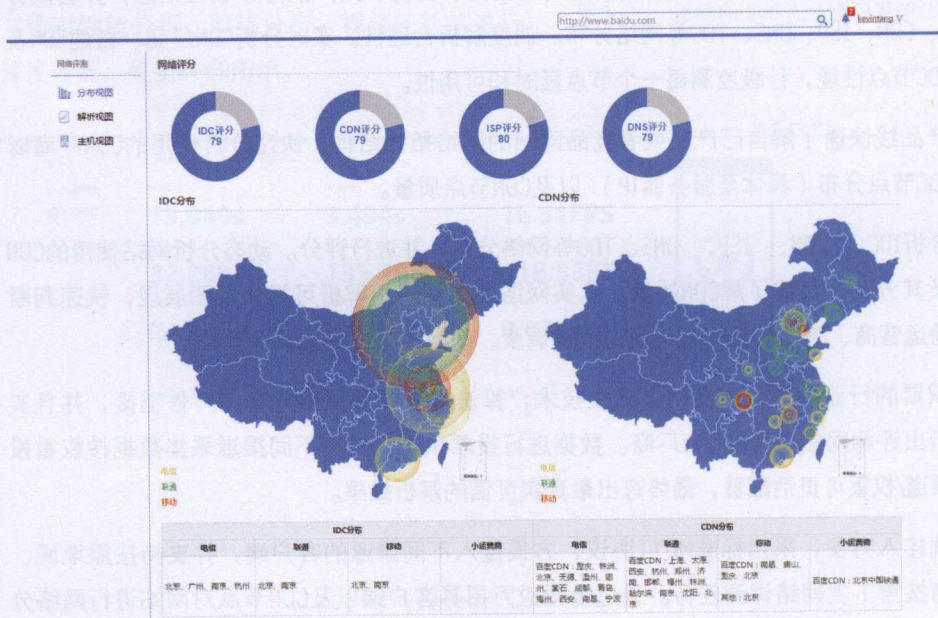


图2-70 分布视图

综合评分通过对IDC、CDN、ISP、DNS四个核心网络属性进行评分，评估网络的合理性。

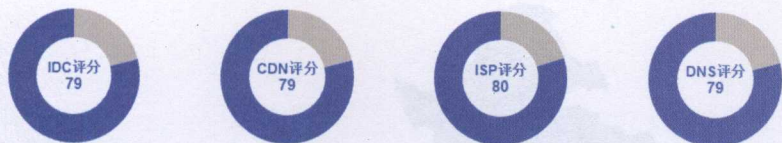


图2-71 评分视图

分布视图详细列出了IDC与CDN节点在不同运营商、不同城市的分布情况，通过全国地图展示IDC与CDN不同的运营商节点在全国的分布与解析情况。可以一眼看出三大运营商的服务器分布情况，CDN节点数量及分布一目了然。

解析视图详细列出了IDC与CDN节点在不同运营商、不同省份的解析情况，有利于分析IDC与CDN节点解析是否合理，是否存在跨网解析情况等。对于有多个镜像站点的源站，管理人员根据覆盖地区和运营商进行了DNS解析设置，通过解释图也可以检验DNS的配置是否正确。例如镜像1在电信负责电信用户接入，通过解释视图发现部分省的联通解析到了镜像1，反映出DNS设置时候可能存在对某些地区运营商的配置错误，或者DNS服务商在该地区的服务有问题，进行定位和排查。

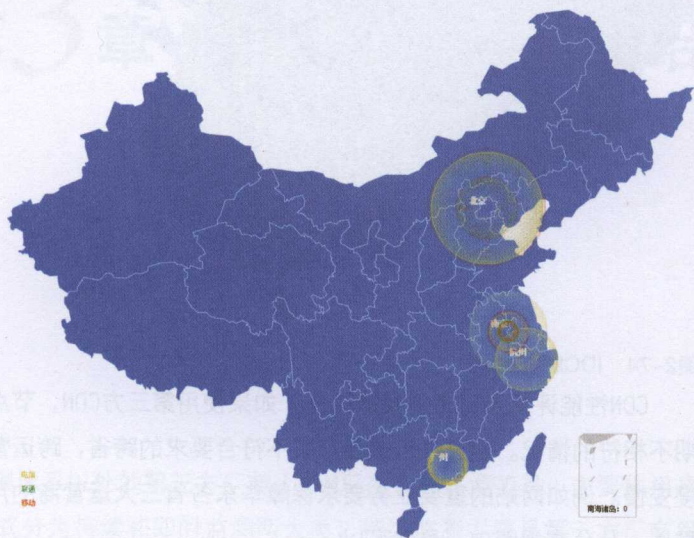


图2-72 IDC分布视图



图2-73 CDN分布视图

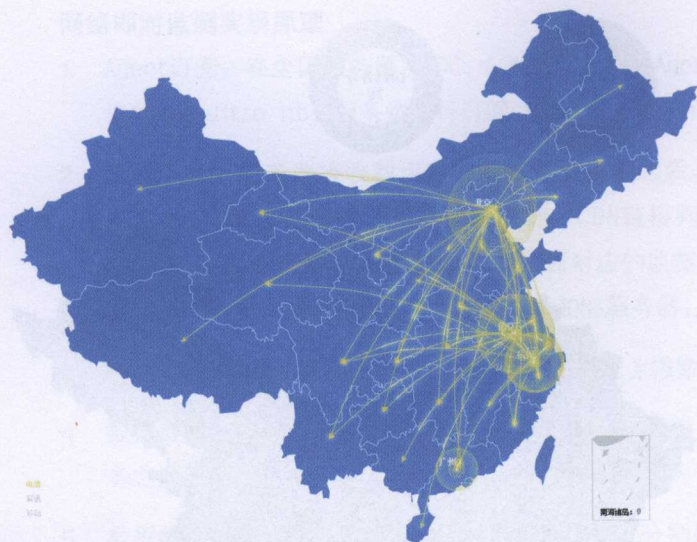


图2-74 IDC解析视图

CDN性能评估是日常重要的工作,如果使用第三方CDN,节点和调度会变化,而且存在很多与预期不相符的情况。通过分析是否存在不符合要求的跨省,跨运营商解析,从而可能导致页面访问速度变慢。例如网站的重要业务要求保障华东各省三大运营商的用户本地覆盖,本运营商覆盖,快速接入。从分布视图中,虽然可以看出各省都有CDN节点,但是有可能CDN厂商有会对解析做了打包解析,这就不符合网站的服务要求。只有通过解析视图列表,可以直观地查询到,各省的解析来源,例如发现浙江电信的用户解析来源于福建电信,说明服务商用邻省节点进行覆盖或者打包解析,那就不符合本省覆盖的要求。如果发现电信的用户解析来源自联通,说明服务商采用联通节点覆盖电信用户,发生跨运营商解析严重影响用户的接入访问体验。

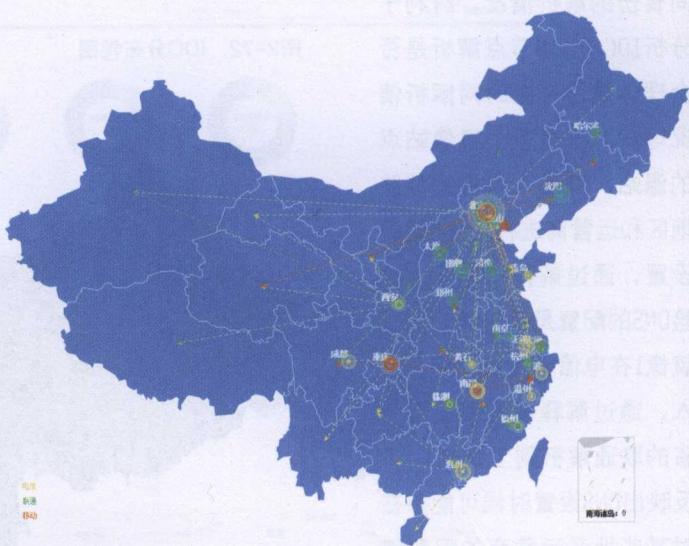


图2-75 CDN解析视图

第3章 性能监测工具介绍

3.1 监测工具概述

这里所说的工具主要指自有监测体系以外的第三方厂商（APM厂商）和开源工具，主要作用是辅助自建性能监测体系，按监测方式分为持续和即时监测两大类，持续监测主要是第三方厂商居多，即时监测多数为开源工具。与自有搭建的监测平台更突出切合企业自有业务特性不同，第三方厂商和开源工具监测粒度更大、更通用，通常也更基础。在建立企业级性能优化的初期，第三方厂商和开源工具的作用是不可替代的，可以快速从0开始建立基础的应用性能数据，而且各家第三方厂商和开源工具因企业背景或团队不同等原因各有擅长的领域，目前主要有以下两种类型：

- 1 EUM, End user monitoring, 最终用户性能监测。通过在最终用户设备上安装客户端的方式，通过主动监测获得采样的性能数据，Keynote是EUM的鼻祖，国内基调、博睿也属于EUM。
- 2 RUM, Real user monitoring, 真实用户性能监测。通过嵌码的方式获得全用户真实性能数据。嵌码方式分三种：网页嵌JS、移动APP嵌SDK、服务器端嵌代理。Newrelic和Appdynamics都支持这三种方式，国内基调、OneAPM也基本支持这三种方式。

为什么传统APM厂商将被淘汰？

传统APM典型产品包括：CA的Wily、IBM的Tivoli、Compuware的Vantage、Oracle的RUEI、HP的APM、OPNet的Xpert系列等，如图3-1所示。性能对比如表3-1所示。网络应用环境已经改变，传统APM没有跟上变化，将被逐步淘汰。

- 1 传统APM解决方案无法满足“虚拟化技术、云计算、新的开发语言、分布式主机环境”下的应用性能监测。
- 2 传统APM产品大多专注在Java和.NET应用程序，对Ruby、PHP、Python、Scala、Node.js等应用支持不足。
- 3 传统APM产品都需要进行手动配置，效率上无法跟上敏捷开发的更新频率。
- 4 传统APM产品很难监测分布在多个数据中心或者公有云上的应用，而分布式部署和云计算的趋势已经越来越明显。
- 5 传统APM解决方案价格昂贵、部署复杂、使用困难。

新型APM厂商有哪些特征和优势？

新型APM典型产品包括：Compuware的Dynatrace、NewRelic、APPDynamics、AppNeta的TraceView、DELL(原Quest)的Foglight、Vmware的vFabric、ExtraHop、Critttercism等，如图3-2所示。新型APM解决方案都有这样的特征和优势：

- 1 支持跨网络的应用性能监测（内部、互联网、云、移动）。
- 2 端到端多层应用性能的可视性。
- 3 深入到代码级（语言和数据库）的性能问题定位。
- 4 网络应用和网络拓扑结构的自动识别，做到“零配置”。
- 5 真实用户体验的实时监测。
- 6 支持更多的开发语言。
- 7 统一的解决方案，而不是多个产品或模块的拼接。
- 8 更简单的部署，更直观的报表，更容易使用和试用，更便宜的价格。



图3-1 传统APM厂商



图3-2 新型APM厂商

表3-1 APM厂商产品对比

公司	EUM产品	RUM产品
Compuware	Gomez	Dynatrace
AppDynamics	与BMC Coradient 合作	AppDynamics

续表

公司	EUM产品	RUM产品
CA	Watchmouse	Wily
NewRelic		RPM
OPNet		Xpert系列产品
BMC	Coradiant	Application Diagnostics, MainView等多个产品
Oracle		RUEI
Ip-Label	newtest	Datametrie
博睿	博睿监测	
基调	RPC (慧眼)	透镜

3.2 持续监测工具

持续监测主要以第三方厂商为主，他们常把自己归属到APM（应用性能管理）行业，APM行业从1995年到现在，经历二十年的时间，但主要历程发生在国外，目前国外这个领域依旧正处于百花齐放的阶段，New Relic和Appdynamics为代表的新秀发展非常迅速，还有很多创业公司不断进入这个领域。APM是一个不断进化和变化的市场，发展到现阶段主要有两个延伸的方向，分别是Mobile和SaaS，Mobile主要随移动互联网发展而变得愈加重要，中小企业的应用性能监控需求促使基于SaaS交付模式的产品更加流行和繁荣。全球领先的信息技术研究与咨询公司Gartner多年跟踪APM行业发展，并对APM厂商做出了以下定义：

- 1 产品必须包括APM五大功能维度(最终用户体验监控、应用拓扑发现与可视化、用户自定义事务处理剖析、应用组件深入监控以及IT运行分析)。不仅能够监控应用，还可以监控一个或更多关键应用组件类型(例如数据库、应用服务器等)。
- 2 产品必须提供生产环境中编译后Java和.NET等代码的运行状态。
- 3 所提供的案例客户至少位于三个区域：北美、南美、EMEA(欧洲、中东及非洲)、亚太地区。
- 4 至少有50个客户正在使用其APM产品。
- 5 所提供的案例客户的应用性能监控对象数目不少于200个。
- 6 产品必须提供基于SaaS的交付模式。
- 7 产品在上一年内的销售额不低于500万美金。

在下面的章节中将重点介绍国内、外优秀APM厂商的产品及特点，以便能帮助大家熟悉和驾驭这些工具，在企业内部应用性能优化中产生价值，第三方服务商及功能对比如表3-2所示。

表3-2 主流第三方服务商及功能

企业/产品	DynaTrace	Newrelic	AppDynamics	TingYun	OneAPM	云智慧
PC、移动、网络评测	✓	✗	✗	✗	✗	✗
移动SDK监测	✓	✓ MOBILE	✓ Mobile Real-User Monitoring	✓ 听云APP	✓ Mi	✓ 移动端真实用户体验
移动JS监测	✗	✓	✓	✗	✓	✗
最终用户端主动监测	✓ 模拟监测	✓	✓ Browser Real-User Monitoring	✓ 听云Network	✓ Bi 类似JS监测	✓
服务器端真实用户监测	✓	✓ BROWSER	✓	✓	✓	✓
可用性监测	✓	✗	✓ Browser Synthetic Monitoring	✗	✓ Cloud test	✓ 网站监控
服务器性能监测	✗	✓ SERVER	✓ Server Monitoring	✓ 听云Sys	✓ Oneapm Servers	✓ 服务器监控
服务器应用代码级性能监测	✓ 应用监测	✓ APM	✓ Application Performance Management	✓ 听云Server	✓ Ai	✓ 应用性能管理
业务流监控	✓	✓	✓ Application Analytics	✗	✗	✓ API皆空
基础设施监测 (硬件+中间件+DB)	✗	✓ PLUGINS (Web+、DB)	✗	✗	✗	✓ 基础设施性能分析(主机+web+DB) docker在测试
数据库监控	✓	✓	✓ Database Monitoring	✗	✗	✓
集中告警平台	✗	✗	✗	✗	✓ Onealert	✗
真实用户环境测试	✗	✓ SYNTHETICS	✗	✗	✗	✓ 压测
私有化交付	✗	✗	✓	✗	✗	✓

3.2.1 Keynote

Keynote是lastmile端到端布点方式监测性能技术的鼻祖，1995年成立于美国并在纳斯达克上市，2013年6月被私募公司以3.69亿美元收购后退市。客户主要为YouTube、Facebook、戴尔、微软等。Keynote虽然没有进入中国，但对中国APM行业的影响深远，Keynote的中文即为“基调”，中国基调网络2007成立以来采用的一直是这种技术，目前已经在新三板上市。Keynote在后期也没有

坚持把主动性能监测产品作为业务重点，而是转向了移动真机测试。目前Keynote已经被Dynatrace收购。Keynote的真机监测，以及真机压测已经归属到Dynatrace产品体系下。Keynote的产品专注移动真机测试和APP真机监测，有着全球最大的云测试，监测和分析网络，每天收集了超过7亿的移动和网站性能监测数据。

Keynote产品介绍见表3-3。

表3-3 Keynote产品及介绍

产品	监测类型	优势	劣势	备注
MOBILE TESTING 移动真机QA测试：通过Keynote提供的真机对APP或者移动Web的质量检测	主动	真实手机测试，机型丰富； 鼠标操作，模拟真实手机触屏操作； 对移动应用可录制脚本测试； 国际主流无线运营商覆盖多	收费按照手机支持类型数量，普通用户收费较贵	通过PC端软件远程操作真实手机测试或自动化测试，可以集中化管理测试手机
MOBILE APP MONITORING 移动APP性能监测：将APP安装在Keynote在各运营商的真机进行实时性能监测	主动	100%基于云的APP真机监测 可以录制脚本	通过获取固定监测点提供的数据，不能完全代表真实用户的性能指标	移动Web&APP监测 可以用Keynote的真机做监测，也可以用APP SDK方式用真实用户监测
SYNTHETIC MONITORING 真机监测：Keynote原有的全球真机点主动监测	主动	最早做真机监测，技术积累比较多。全球分布2500多真实用户监测点，覆盖240个地区和160个城市	已跳转到Dynatrace Synthetic Monitoring	Web真机监测，已被Dynatrace收购，与Gomez是一类产品，是海外竞品性能对比的选择之一，例如之前在百度以测试海外baidu vs google检索时间差异就使用了Keynote
DIGITAL PERFORMANCE INTELLIGENCE 行业对比性能分析：从性能角度进行行业对比分析，基于Keynote的监测数据以及第三方数据	数据分析	将各个测试模块的数据以及第三方数据（Google Analytics, Cedexis等）进行大数据整合展现	暂无	主要面向客户做竞品对比
LOAD TESTING 真机压力测试：基于监测点的真实访问压力测试	主动	对Web和移动产品的云压力测试工具	已跳转到Dynatrace Load	Keynote Load Testing 已被收购到Dynatrace load

3.2.2 Dynatrace

Dynatrace 2007年成立于美国，2011年被Compuware收购。2014年，Compuware被私募股权投资

公司Thoma Bravo收购后，Dynatrace被再度分拆为一家专注于APM的独立公司，1600员工，目前是APM领域销售额第一的公司，年销售额接近4亿美元。2014年营收增长率超过20%，达3.269亿美元，占据12.5%的市场份额，连续5年被Garner魔力象限评为APM市场的领导者；Dynatrace客户5800家，其中386家世界500强企业。十大银行和十大零售商中，分别有九家使用了Dynatrace产品。无论是销售政策和渠道执行力，还是新技术的应用和商业合作的拓展，Dynatrace的转型是相当快速和成功的。Dynatrace 推出了一整套与其公司同名的应用性能管理(APM)产品组合，包括提供应用深入分析的Dynatrace 应用监测，用户深入分析的Dynatrace 用户体验管理和Dynatrace 模拟监测，应用感知网络深入分析的Dynatrace 数据中心真实用户监测。

全新的产品系列名称和品牌强化了以用户为中心的新一代 APM 产品的集成性和统一性，从终端用户视角了解应用和数字化渠道所需的关键深入分析，为用户提供综合的应用性能管理服务。这些新产品的上线，以及定位于中小企业的SaaS服务的推出，虽然为Dynatrace带来的收入增长并不太多，但却把企业拓展到了远超传统客户数量的新兴市场中。以Dynatrace的技术基础和市场积累，预计在2016年仍然会继续保持并巩固其APM领军地位。

Dynatrace产品介绍见表3-4。

表3-4 Dynatrace产品及介绍

产品	监测类型	优势	劣势	备注
Dynatrace Application Monitoring 应用监测：通知安装探针方式，对应用进行性能监测	探针	跟踪事务，从点击到产生的方法调用直至在基础设施内部执行的SQL语句； 直观地了解所有服务、层、相互依赖性和层间时序的实时逻辑拓扑	首页上只提供Java，.NET，PHP三种语言	从用户到代码，的端到端业务性能监测，视图丰富
Dynatrace Load 真机压力测试：通过真机产生访问压力	主动	通过真实用户点产生压力访问，最大1百万并发用户	可能采用的也是JS调用真实网站访问用户去做测试。 从Keynote收购	与AkamaiCDN是合作伙伴
Dynatrace User Experience Management 用户体验管理 在统一视图中比较和分析用户体验，涵盖了网络，应用，后端数据库等多个层面的监测数据聚合	主动/被动	对Web，移动Web，APP监测聚合到统一视图，定性和定量的分析客户行为； 在统一视图中比较和分析用户体验，可覆盖从应用到后端的任何渠道； 从应用到后端全面捕获信息，自用户单击开始，接下来是涵盖所有层、第三方和云服务的交易，直至记录和备份数据库	偏重于移动应用	原产品名DynaTrace Real User Monitoring for Mobile & Web

续表

产品	监测类型	优势	劣势	备注
Dynatrace Synthetic Monitoring 真机监测：利用节点和真机进行性能监测	主动	覆盖168个国家和地区终端PC； 150个骨干网节点； 70多个3G和4G运营商网络	收购自Keynote	原产品名APMaaS Synthetic Monitoring for Mobile & Web, Gomez
Dynatrace Data Center RUM 数据中心真实用户监测： 通过单一仪表板，从关联的视角了解企业应用和网络的健康状况	聚合	从同一个视角对用户体验、应用和网络性能进行分析，找到并解决问题。 对所有的应用、所有的层进行实时分析。覆盖数据库、中间件应用和服务，结合深入的交易性能分析	暂无	以事务流程为中心的性能监测； 可以根据性能、可用性和使用情况来管理整个企业应用组合。 可针对各种不同类型的数据库、中间件应用和服务进行深度交易分析，包括IBM MQ、XML/SOAP、Oracle、DB2 Sybase、MSSQL、LDAP、SMB/CIFS、Web服务器、Exchange等等

3.2.3 App dynamics

2008年成立于美国，通过服务器端嵌代理的方式监测应用性能。优势是关键业务过程性能可视化及网络路径的自动识别跟踪。客户主要是中大型企业。Appdynamics和Newrelic的CEO都出自Wily。Gartner给了Appdynamics更高的评价。Appdynamics走的是大客户线路，主要为企业内部应用提供定制化的应用性能管理服务。这种模式在中国目前更加有市场，因为中国的SaaS化程度20%都不到。而且大客户将带来更高的客单价，不足的是，项目制将增加人员成本，用户基数不足。Gartner在APM 2015年的魔力象限报告中，给了Appdynamics很高的评价，并将其定位为APM行业的领军企业。Appdynamics也在2015年第四季度发布了16项更新，加强了对用户行为、会话、应用基础设施、浏览器主动监测以及C/C++的应用监测支持，为用户提供更丰富，更有深度的数据视图，使应用性能管理和企业商务运营具有更强的关联性。其发展方向从基础监测，向数据整合，数据分析延伸，技术实力强，更新非常快。

App dynamics产品介绍见表3-5。

表3-5 App dynamics产品及介绍

产品	监测类型	优势	劣势	备注
Application Performance Management 应用性能监测：服务器端植入监测探针，对应用性能监测	探针	支持C++(beta); 获得跨越应用程序拓扑的实际事务的集成视图; 在任何一个层或节点，钻取每一个交易，发现，诊断问题; 捕获和相关的实时业务指标的应用性能; 支持拖放HTML5的仪表板	未见提供Ruby	应用性能监测
Mobile Real-User Monitoring 移动用户监测：对APP植入SDK，采集APP使用过程中的用户性能和崩溃	SDK	APP 监测 可以和应用监测进行top视图结合，视图可自定义	暂无	APP SDK监测
Browser Real-User Monitoring Web监测：基于JS的页面性能监测和分析	被动	常规的被动监测指标，页面加载时间，Ajax，加载错误等	暂无	JS被动监测
Database Monitoring 数据库监测：通过设置数据库账号，对数据库性能数据进行轮询采集以及可视化展现	主动	DB2, Oracle, SQL Server, Sybase ASE, MySQL, Sybase IQ, PostgreSQL, MongoDB, and NetApp 额外开销小于1%	需要提供数据库访问权限的账号，至少包括select, process和show databases	通过有一定权限的数据库账号来查询数据库性能
Server Monitoring 主机监控：服务器安装监测探针，采集服务器运行时的性能数据	探针	主要监控linux和windows主机的CPU, MEM, 磁盘，网络; 产品支持Docker和VMs	有部分功能是beta版	在beta版中有数据中心->机架的逻辑分级结构
Application Analytics 应用分析：将监测平台数据进行聚合，按客户所需进行可视化展现	数据聚合	将业务数据和多种监测数据进行了关联 自定义分析界面，自定义界面	暂无	大数据平台进行实时的业务和应用性能的分析
Browser Synthetic Monitoring (Beta) 真机监测：从监测节点发起主动探测	主动	监测终端分布在全球的23个数据中心	提供Beta版	应该是从全球的节点发起访问，不是最终用户

3.2.4 Newrelic

NewRelic 2008年成立于美国，定位为部署在数据中心或者云端、移动应用提供 “All in one” 的应用性能管理服务供应商，公司CEO兼创始人是Lewis Cirne，是美国性能监测领域的先驱者，

Circle于1998年创办了Wily Technology公司，于2006年将Wily出售给了全球最大IT管理软件公司之一CA Technologies。Newrelic产品专注在SAAS服务上，客户主要是中小企业、初创企业，拥有更广泛的用户基数。从长远来看，SaaS模式必定是趋势，但在短期，大客户的服务能快速增加收入。

在2015年12月的Gartner APM魔力象限报告中，Newrelic持续4年被评为APM魔力象限的领导者。Newrelic基于SaaS的业务模式，为其带来巨大的影响力和快速的增长速度。Newrelic很早便意识APM的采购者会越来越弱化专门的IT运维部门这一趋势，因此在产品方案上，其设定目标是在企业的研发，运维，运营之间构建大数据分析渠道。目前看Newrelic的产品系类已经比较丰富，从大量的监测数据中，进一步向数据聚合和数据分析拓展，特别是Newrelic的Insights。Newrelic仍然会继续SaaS路线，保持对功能的快速更新，定位自己为软件分析的先锋。

Newrelic产品介绍见表3-6。

表3-6 Newrelic产品及介绍

产品	监测类型	优势	劣势	备注
New Relic APM 应用监测：通过植入探针方式监测后台应用的响应时间性能，并定位到代码分析	探针	Ruby, PHP, Java, .Net Python, Node.js 监测功能比较丰富 拓扑图可拖拽和自定义	未见C++	主要监测对应用的响应时间，慢响应事务，数据库调用的性能，对关键事务的定义和性能告警
New Relic INSIGHTS 从Newrelic的其他监测产品获取监测数据，进行整合和可视化展现	聚合	将应用监测，浏览器监测，移动监测，真机监测数据进行聚合，以及可视化展现 用户根据自身需要进行自定义dashboard	需要首先使用其他配套监测产品	数据来源于其他监测产品，采用NRQL语言实现数据的可视化分析和实时展现
New Relic MOBILE 移动监测：对APP植入SDK进行性能监测	SDK	采集，HTTP时间，错误，崩溃，以及地理，运营商，版本等信息，基本上都是目前APP监控的主流指标	未见TCP网络层指标	主要还是基于Web view的性能指标采集
New Relic BROWSER 浏览器监测：通过JS方式对页面前端性能监控	被动	JS监测 有采集AJAX指标 有Session Trace 对加载过程中的AJAX请求，用户交互，JS执行和错误查看	JS脚本嵌入	JS在客户端下载后会持续一段时间的采集，因此采集数据比较多，有助分析页面加载瓶颈
New Relic SYNTHETICS 真机监控：真机模拟用户主动访问监测	主动	全球布点真机访问 网站加载时间和可用性，API监测 可记录错误截屏。 有SLA定义告警规则 在产品页面上有一键评测的功能，可任意评测某个URL，免费次数有限制	监测点地区集中在 Washington DC, Portland, San Francisco, São Paulo, Dublin, Singapore, Tokyo, and Sydney. 基于Chrome 浏览器	主要采用瀑布图的方式展现监测时间指标

续表

产品	监测类型	优势	劣势	备注
New Relic Server 主机监测：主机上安装探针，采集物理设备性能指标	探针	支持Redhat、Ubuntu、Debian、CentOS Windows、SmartOS Amazon EC2 支持Docker 可提供API查询接口	未见有内外网混布情况的说明	比较常规的服务器性能监控。可以和APM以及Broswer进行数据整合
New Relic Plugin 插件监控：通过安装Meetme agent 或者利用数据库的admin账号，主动采集中间件或者数据库的性能，将数据可视化展现	主动	对中间件基础平台的监控如数据库，HTTP server，以及插件的监测并且支持用户自定义插件监测	对数据监控需要开通账号	利用各种平台或者插件自带的性能查询接口，进行采集并将数据可视化展现
New Relic ALERTS 集中告警平台：将其其他产品平台的告警集中整合	告警	告警平台	暂无	和所有监测平台对接，提供集中告警
New Relic for iOS & Android 客户端APP，可以查看Newrelic的各产品检测数据	客户端APP	提供性能监测的手机端APP	暂无	通过APP查看监测数据，可支持Newrelic的APM，Mobile，Server 和Browser

3.2.5 基调

基调网络于2007年成立，是国内首家从事应用性能管理和用户体验优化的第三方监测服务提供商，是技术导向的企业，总部位于北京，在上海、广州、深圳、成都分别设有分公司，2015年6月已经在新三板上市。技术持续投入、成熟的销售团队、完善的售后服务使基调保持行业领先优势。市场份额约55%~60%，SaaS听云平台注册用户超3万开发者及中小用户，2000+企业级用户。2015年底的Gartner APM 魔力象限报告，听云作为中国厂商入选，虽然排名并不是很好，但是表明其在上市后，对市场形象，国际化影响有更多的投入，以期后续更好的资本运作。在2015年圣诞节推出了iDaas业务，也说明听云在向大数据分析方向延伸。

基调产品介绍见表3-7。

表3-7 基调产品及功能

产品	监测类型	优势	劣势	备注
听云Network 真机监测：LM监测点主动访问的监测	主动	真机监测，lastmile点比较稳定覆盖的地区和运营商比较全 Web监测的配置选择比较多，功能较全 有个即时监测功能，能调用监测点临时做一次访问	视图比较老旧 监测点都是IE浏览器 首屏仍然采用800*600 Web监测没有白屏时间	产品已经比较成熟，并且在市面上引用较多，多用于对CDN效果的评估。实际上主要客户还是用原来的Network老版本，用户界面已经过时
听云APP 移动监测：对移动APP植入监测SDK后的性能监测	移动 SDK 监测	11月新增了APP对TCP网络层时间的获取 有劫持分析 Android SDK 1. 新增SSL时间，首包时间，c的socket的建连时间，Webview中的DNS，SSL时间，建连时间 对Gradle 1.5的支持	没有城市级别视图	APP监测也是今年才开始推出，更新迭代速度很快
听云Server 应用监测：通过在服务端应用中植入探针，实现对应用代码、关系型数据库、NoSQL、外部服务、服务器本身的监控	探针	应用拓扑视图可拖拽自定义 关键应用过程监测 有线程剖析功能 后台任务展现 提供了应用环境信息展现	没有Ruby探针 Node.js和Python探针还是Beta版本	SDK基本上1个月更新一次版本
听云Browser 页面监测：基于页面JS嵌入技术的真实用户性能监测	被动	有AJAX请求统计 JS错误统计 慢页面追踪 浏览器视图中IE可细化到版本	缺少网络层时间指标 目前是Beta版	JS被动监测（尚未开通） 可以监测页面加载时间、JS错误、AJAX请求等性能指标，为用户的HTML5页面，微信公众平台号，混合应用等网站提供准确和详细的真实用户体验评估，定位高并发下的网站性能瓶颈
听云Sys 主机监测：在服务器上部署探针，实时获取服务器的各项指标数据，如内存、CPU、磁盘、进程对资源的消耗、操作系统参数等	探针	支持安装类型和机型比较全 RPM/DEB/BIN/EXE安装包 RedHat 5.0 / CentOS 5.0/ Suse Linux 10.0/ Debian 5.0 / Ubuntu 9.10/Windows操作系统	免费账号最小时间粒度到30分钟 同时最多10个服务器 未见有内网解决方案	比较常规的主机监测，突出对服务器的额外开销较小 应用延时<1ms CPU使用<1% 内存使用<100MB 带宽消耗<1MB

续表

产品	监测类型	优势	劣势	备注
听云 iDaas 基于大数据的性能分析	整合	聚合行业性能数据的对比分析，将行业性能数据可视化，结合用户场景的多维分析	Beta版	从横向对比来引导用户性能监测
听云警报：集中告警平台	聚合	将APP，Server，Sys的告警信息进行聚合展现	Beta版	近期刚刚推出

3.2.6 博睿

博睿于2007年在北京成立，从最初只提供网络监测服务，逐渐转型做APM市场，产品也在逐渐丰富，现在线上产品绝大部分为主动式真机监测。2015年博睿的转型变化非常明显，对首页进行多次改版升级，对产品也进行重新规划命名，陆续推出移动APP监测、Mobile RealAPP Reeiss等产品。从其最新的产品架构上来看，向移动应用偏重较多，并且推出了APP的功能测试 APP Testing产品，因此对移动APP，目前其产品可以覆盖从功能测试到上线后性能管理。

博睿产品介绍见表3-8。

表3-8 博睿产品及功能

产品	监测类型	优势	劣势	备注
Bonree Synthetics Monitoring 博睿用户体验监控：利用Bonree多维度均匀分布的真实监测点（IDC和LM），主动探测，采集用户感知数据。	主动	真机监测是其传统业务	界面风格比较老	10万LM用户130个IDC节点，竞品分析，IDC评估，CDN评估，SLA，NOC监控，劫持监测等
Bonree Browser Real-user Management 博睿用户体验管理：网页中嵌入JS，实时监测，实现对所有用户的响应监控	被动	HTML5页面 微信平台公众号 混合应用 自定义提取性能指标定义	暂无	在Web网页中嵌入JS，实现对所有用户的响应监控
Bonree Mobile Synthetics Monitoring 博睿APP性能监控：通过各地多家移动运营商的真实手机监测点，连续采集手机4G/3G/2G接入条件下的移动APPs应用性能	主动	移动APP真机监测 最终用户视角应用性能监测 支持云部署的应用性能追踪 追踪错误成因定位性能问题 应用启动互动流程性能结构 真机多维度组合分析	暂无	通过各地多家移动运营商的真实手机监测点，连续采集手机4G/3G/2G接入条件下的移动APPs应用性能

续表

产品	监测类型	优势	劣势	备注
Bonree OTT Real-user Management 博睿智能硬件体验管理: 面向机顶盒及物联网智能终端嵌入代码收集相关性能数据, 进行用户感知分析	嵌入式	监测从一个物联网设备到另一个物联网设备切换过程中, 应用的用户体验及延续性能	还未上线	面向机顶盒及物联网智能终端嵌入代码收集相关性能数据, 进行用户感知分析的解决方案。针对物联网第三方衔接插件的衔接性能、网络切换、端口切换性能进行监测
Bonree APP Testing 博睿APP云适配: 对象化测试脚本录制, APP在不同手机上同时做遍历型功能深度适配测试	真机	对象化测试脚本录制, APP在不同手机上同时做遍历型功能深度适配测试支持深度遍历型测试	暂无	针对APP的功能, 兼容性的真机QA测试
Bonree Mobile Real-user Management 博睿APP体验管理: 面向APP嵌入SDK代码方式收集相关性能及行为数据进而做交叉分析	SDK	APP SDK监测, 除应用层性能外, 宣称定位网络层, 支持APP的TCP层时间指标获取	暂无	APP真实用户全量监测解决方案, 面向APP嵌入SDK代码方式收集相关性能及行为数据进而做交叉分析
睿思自主云平台5.0: 网站性能监测软件, 可以用云端节点或者自己的节点进行网页监测	PC终端	完全自主化的监测云平台, 在自己搭建的云平台上部署自己的监测任务	暂无	由本地端的页面优化工具转变为监测云平台, 并且增加的自动报警机制

3.2.7 OneAPM

OneAPM前身叫蓝海讯通, 成立于2008年北京, 原来定位主要做企业级信息系统自动化运维领域, 主要产品是blueware TPM。2013年以来获得经纬中国多轮投资后转型定位作为比肩国际的应用性能管理云解决方案提供商, 全新产品OneAPM主要借鉴NewRelic和Appdynamics。从传统软件公司转型过来, 技术基础比较扎实, 产品偏重于服务端应用监测, 目前没有真机主动监测, 整体解决方案还在快速建立中。2015年推出了安全系列产品和告警平台, 在产品种类上虽然显得丰富, 但是在每种产品的完善度上并未太大改进, 产品铺得比较广并不断通过资本运作, 加速产品、技术、市场完善。

OneAPM产品介绍见表3-9。

表3-9 OneAPM产品及功能

产品	监测类型	优势	劣势	备注
Application Insight 应用性能管理：通过在服务端应用中植入探针，实现对应用监测，定位慢响应到代码级	探针	拓扑图可拖拽 有后台任务展示 Agent可自行配置 关键事务自定义	UI页面显得比较杂乱 无C++	常规应用监测
Browser Insight 浏览器监测：基于JS的Web 前端真实 用户性能监控，统计分析网站流量，定位网站性能瓶颈	被动	有快照功能 对监测对象区分访问域名和访问页面 有AJAX性能 脚本错误视图 支持浏览器、微信、App 浏览 HTML 和 HTML5 页面	没有定位到城市	基于真实用户的 Web 前端性能监控平台 定位网站性能瓶颈，网站加速效果可视化
Cloud Insight 系统监控和中间件监控：集监控、管理、协作、计算、可视化于一体的监控数据可视化展现	探针	对监控指标进行了聚合 Cloud Insight 支持多种操作系统和云主机的监控，在一个平台上对所有基础设施进行集中管理。通过标签，对基础设施进行有效地管理	从页面上看应该还处于初步阶段	集操作系统监控（如 Ubuntu, CentOS, RedHat 等），和云主机监控（如 Amazon Linux），以及数据库监控（如 MySQL, MongoDB 等），和中间件监控（如 Tomcat, ActiveMQ 等）于一身
Cloud Test 云拨测：通过节点主动探测，对页面可用性，PING，API进行实时监控	主动	可用性监测，响应时间监测，API接口监测 只记录响应时间，有省份，运营商维度 有散点瀑布图，可看页面加载项	应该是从节点发起的拨测，比单纯的可用性监测信息更细，但是比普通的Web真机监测信息量少	可用于网站/页面的可用性监测
Mobile Insight 移动应用性能监控：对移动APP植入SDK，采集APP上线后真实用户使用过程中的性能和崩溃	SDK	有个用户信息功能，可根据手机的IMSI关注某一台手机的崩溃信息 ANR信息显示 卡顿信息显示（Beta） Socket 监控（Beta） 组合分析	有个用户信息功能，可根据手机的IMSI关注某一台手机的崩溃信息 ANR信息显示 卡顿信息显示（Beta） Socket监控（Beta） 组合分析，没有网络TCP时间，部分功能只在Android上有，iOS没有体现	比较常规的APP SDK 监测

续表

产品	监测类型	优势	劣势	备注
OneAPM Server 主机监控：服务器端安装 探针采集服务器性能指标	探针	CPU，网络，磁盘，进程监测	没有列入oneapm的主产品。	提供对服务器资源状态监控的轻量级服务，用于和Ai一起查看应用的性能和所在 Server 的资源状态
ONE ASP 自适应安全平台：提供 给企业一个全面且具有 高集成性的自适应安全 平台，ASP (Adaptive Security Platform)，具 有预测 (predictive)、 预防 (preventive)、检 测 (detective) 和响应 (responsive) 的能力。 ASP启用一种新的信息安 全体系 (ISA)——把重 心从“事件响应”转移到 “连续响应”。力求全面 且持续的监测，从而保证 系统安全 能够提供 SaaS 模式和本 地化部署模式	安装部署	RASP，将防护行为注入应用 程序和外部通讯的节点，通 过探针API对这些注入点进行 监控	产品是最近推出	四类产品 (OneRASP、 OneSEP、OneSSS、 OneSIEM)，覆盖对系统 安全的大部分需求。 OneRASP的安全保护控制 点通常放在应用程序和 其他系统的交互连接点 上，包括和用户、数据 库、网络以及文件系统 的连接点。OneRASP在 这些节点上监听所有交 互行为，一旦发现威胁 行为，在监控模式下， OneRASP会记录威胁的攻 击路径，提供如何修复 这些问题的建议并通知 安全管理员。在防护模 式下可以实时拦截威胁 行为
Onealert 报警管理平台：将各平台 的告警事件聚合，流程化 处理和分析	聚合	分布式监控，集中化告警 OneAlert 在一个平台中接收 所有监控，支撑系统事件， 全面掌握所有IT重要事件 自动升级处理，告警统计分 析 支持短信，微信，电话，APP (开发中)	新推出	统一化告警平台，和其 他客户交流中，据说可 以和用户自己的告警平 台对接
.Log Insight 日志采集分析平台：日志 集中上传，可视化分析	聚合	使用现有系统，将任何类型 的日志实时上传到LogInsight 中 支持Agent和非Agent接入。 基于Grok的日志解析规则， 通过交互式界面抽取并结构 日志	刚推出，界面不是很 友好	对日志的采集、存储、 搜索、可视化、告警

3.2.8 云智慧

云智慧于2009年在北京成立，定位为基于SaaS的APM服务商，未来将向企业大数据服务商的角色转变。值得一说的是云智慧的前身监控宝，监控宝是国内最早尝试IDC到业务端可用性监控的SaaS服务商，被云智慧收购后开始转型进入APM行业，正因监控宝切入较早，而且对小微用户免费，积累下大量小微客户。2015年推出透视宝（提供面向移动、浏览器、主机、应用的监控与管理解决方案）和压测宝。而云智慧是对三者基础功能进行整合包装，打造统一平台的监控、管理和测试的产品体系，从云智慧的产品划分上看，仍然是监控宝为主的比较基础性能监测，真正的APM应用性能监测透视宝也是刚刚开始发力。其APM产品划分也是学习了Newrelic和Appdynamics，云智慧仅在概念上进行了组合包装，产品层面割裂比较明显。在获得B+融资以后，后续应该会对产品线进一步的整合。

云智慧产品介绍见表3-10。

表3-10 云智慧功能及介绍

产品	监测类型	优势	劣势	备注
应用监控:在服务器应用端植入探针监控应用性能，定位慢响应和代码分析	聚合	以业务为单位对监控项目进行统一管理和展示实现全栈可视化分析面向系统级的IT监控规划	只是个概念包装，具体功能是由监控宝和透视宝实现	基于业务视角全局展现应用系统的所有技术栈及其关联关系，帮助运维确定企业核心业务组成与优先级，确保系统的SLA，真正让运维引导及支撑业务运行
网站监控：从监测节点根据所需监测协议，发起主动探测	主动	监控点150+节点监测 支持协议比较多HTTP/HTTPS、PING、FTP、DNS、TCP/UDP、SMTP、TRACEROUT 对内网主机有内网采集器	节点监测	主用于稳定性和可用性实时追踪 洞察CDN效果DNS状态 全网全地域性能趋势分析
API监控：从监测节点根据所需监测的接口请求方式，发起主动探测	主动	支持GET、POST、PUT、DELETE、HEAD、OPTIONS 六种请求方式 支持移动真机监测点	节点监测	基于API请求快速创建用户操作流程。 分析用户每个操作步骤的API调用情况
服务器性能监控：服务器端开启SNMP client，从平台定期发起SNMP轮询采集指标	SNMP	采用SNMP轮询方式，不在被监测目标安装探针 支持机型丰富，并且支持网络设备，如Cisco路由器 对内网或者混布环境有提供代理采集器的方式支持	SNMP扩展性差，同样需要每台被监测目标开启SNMP服务	基于SNMP的指标采集，类似早期的网管平台

续表

产品	监测类型	优势	劣势	备注
中间件监控：通过中间件的监控接口进行主动轮询采集性能数据，可视化展现	主动	覆盖主流中间件监控 网站服务器，邮件服务器，和消息服务器	应该是利用中间件自带的管理接口或者模块，进行信息采集和汇总展示	对服务器并发连接数，吞吐率、可用率等，重点是告警
数据库监控：通过设置数据库账号进行数据库性能参数的轮询采集，可视化展现	主动	MySQL, MongoDB, Redis, Memcache, SQLserver 在数据库中给监控专门创建账号进行查询	定期通过监测账号查询数据库性能。	监控数据库运行时的各项性能数据，包括查询吞吐率、查询缓存、索引缓存、并发连接、流量以及表锁定等性能报表和分析报告
网页性能管理：从节点发起主动访问，提供网站全景分析监控	主动	涵盖所有6大类23小项网页元素，识别页面元素正确性、响应时间及实时状态 网页是否可用、响应时间是否超过阈值等性能实时监控	非最终用户	IDC节点监测
Docker 监控：服务器端安装Docker监测插件采集Docker运行时的性能数据	主动	实时监控Docker容器的CPU、内存、网络流量及Swap状态	安装SendProxy或者Linux Docker插件	类似主机监测的性能指标
应用性能管理：服务器应用端植入监控探针，监控应用性能	探针	采用了统一的Smart Agent（安装后即可进行主机监控），对PHP, Java, Python等应用通过插件的方式来安装和配置	安装Smart Agent 无C++	常规应用性能监测
移动端真实用户体验：移动APP植入SDK，对APP性能，崩溃进行监测	SDK	移动应用的用户行为、HTTP请求、崩溃、地域、设备等性能分析 iOS版本有行为动作分析 增加流程自定义和分析	无网络层时间	常规
浏览器端真实用户体验：基于页面植入JS的Web端真实用户体验监控与分析	被动	还是需要下载和安装Smart Agent 发现应用之后会自动将一小段JS代码注入到被监控的页面	仍然需要安装Smart Agent	针对Web前端页面的性能分析，包括页面在不同区域、不同浏览器中的表现。按页面进行性能及错误分析并细化追踪单个页面在具体用户访问时的详细信息
基础设施性能分析：聚合主机，数据库，中间件监测	探针	主机性能监控管理包括针对服务器系统运行状态相关的各项性能指标、部署在服务器上的各种中间件（服务）的各项性能指标以及数据库各项性能指标的监控	仍然需要安装Smart Agent	把主机，Server和数据库监测集成到了一起

续表

产品	监测类型	优势	劣势	备注
真实用户压力测试： 通过全球分布式网络 发起真实压力，发现 全链路性能瓶颈	主动	从IDC机房发起压力测试 宣传有500个核心机房服务器， 覆盖150个城市 支持用户行为录制、编辑、回放； 支持技术包括，AJAX， REST， SOAP，HTML5， Silverlight	缺少真实用户压 测能力	可配合透视宝和监控宝，进 行性能压测

3.3 即时监测工具

即时监测主要以开源工具为主，这类工具最大的优势是专注性能而且是一次性快速评测并出结果，与自建监测体系有较大的互补性。这类工具偏前端、移动研发及测试使用，运维、系统工程师使用有一定门槛。而且慢慢由前端向移动方向演变，后端也涉及一部分，但权重较小。当前主流即时性能评测类工具都较分散，因不像第三方厂商或自建监测平台有持续团队维护，开源即时监测工具在发展过程中多少都出现过“断档”现象，而且在深度和广度上都不够。目前较常见的做法是在不同的开源即时监测工具的基础上进行二次开发，加入自有需求并集成到自有监测平台，成为产品研发生命周期中的一部分，在开发、测试过程中杜绝性能问题，在测试和发布前可以通过即时监测将质量问题轻松暴露出来，进而保障发布的内容是最佳用户体验。

3.3.1 YSlow

YSlow(why slow)是雅虎基于网站优化规则推出的网站性能分析工具，能够帮助用户分析并优化网站性能。雅虎网站优化规则在多个方面给你的网站提出优化建议，包括尽可能减少HTTP的请求数、使用Gzip压缩、将CSS样式放在页面的上方、将脚本移动到底部、减少DNS查询等数十条规则，YSlow会根据这些规则分析你的网站，并给出评级，YSlow界面如图3-3所示，基本功能如下。

- 1 系统预定义了三套规则集，用户可以选择其中一套或者使用自定义规则集进行页面评分，三套规则集分别是YSlow(V2)、Classic(V1)和针对小站点的“Small site or blog”规则集。
- 2 向用户提供页面优化建议。例如，使用CDN、设置足够的缓存时间、JS/CSS放置在页面的合适位置、减少DNS解析的次数、对JS、CSS进行Minify处理等。同时，YSlow会列出不符合这些建议的具体请求，并标明这些请求的问题是什么。
- 3 对页面元素进行归纳整理，并给出每个元素详细的性能相关信息。比如，资源原始大小、Gzip后资源大小、发送Cookie大小、接收Cookie大小、HTTP头部信息、缓存过期时间、响

应时间、ETAG值等。

- 统计所有页面元素在有Cache和无Cache两种状态下的体积占比情况，并给出统计图。
- 提供包括Smush.it和JSLint在内的第三方性能分析优化工具支持。
- 除支持Chrome、Firefox、Opera和Safari等主流浏览器外，还支持PhantomJS、Node.js和命令行等服务端工具。

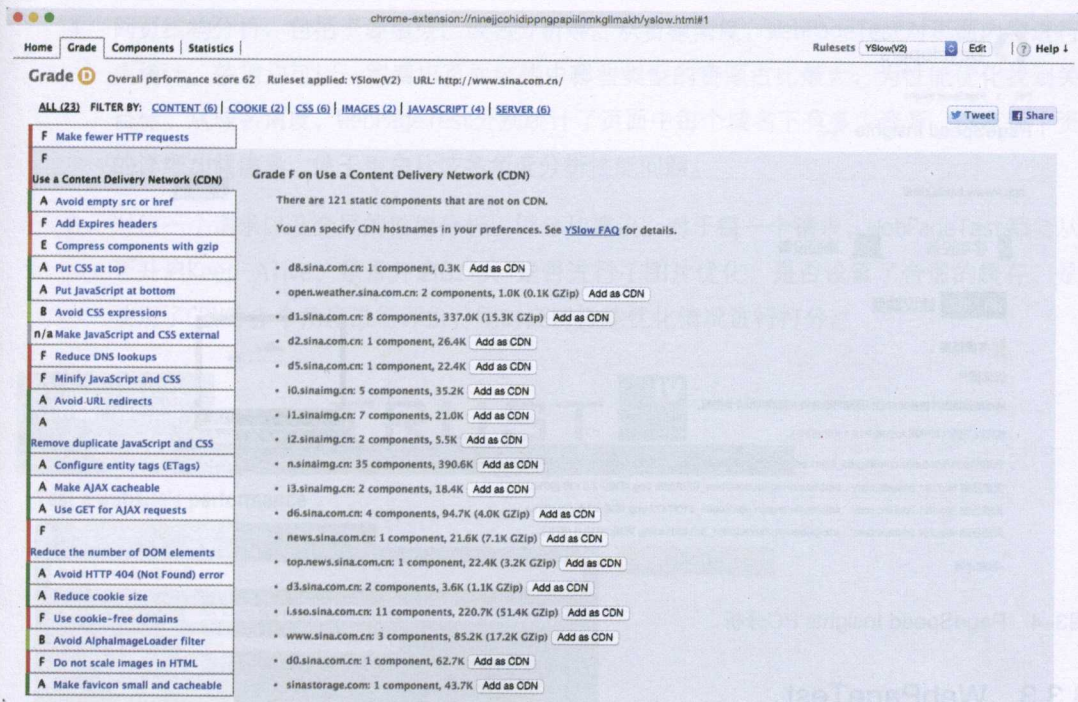


图3-3 YSlow分析视图

3.3.2 Pagespeed Insights

PageSpeed Insights是由Google提供的一套帮助开发者提高网站性能的工具，从而帮助能够带来最佳的渲染性能，尤其针对移动页面，对网页进行分析并提出优化建议，还提供网站优化相关的类库供开发者使用，PageSpeed Insights界面如图3-4所示，基本功能如下。

- 同时在PC和移动终端对网页进行评测。越来越多的网站都会同时提供PC站点和移动站点，性能优化要同时考虑这两种站点，PageSpeed Insights分别针对桌面设备和移动设备同时进行测试，测试结果分为两个标签页展示。

- 2 根据网页性能优化规则，对被测页面进行分析，并提出优化建议。建议中包括浏览器缓存使用、HTML/CSS/JavaScript等文本内容的Minify、启用压缩、按照优先级排列可见内容、清除首屏中阻止展现的JS和CSS、缩短服务器响应时间以及避免重定向等。
- 3 针对移动设备，PageSpeed Insights提出了适合移动体验的优化规则，比如：a.网页上的按钮不能太小，之间的距离不能太近，降低触屏操作时误点的概率；b.使用清晰的字体；c.配置合适的viewport；d.调整内容尺寸使其符合视口设置等。

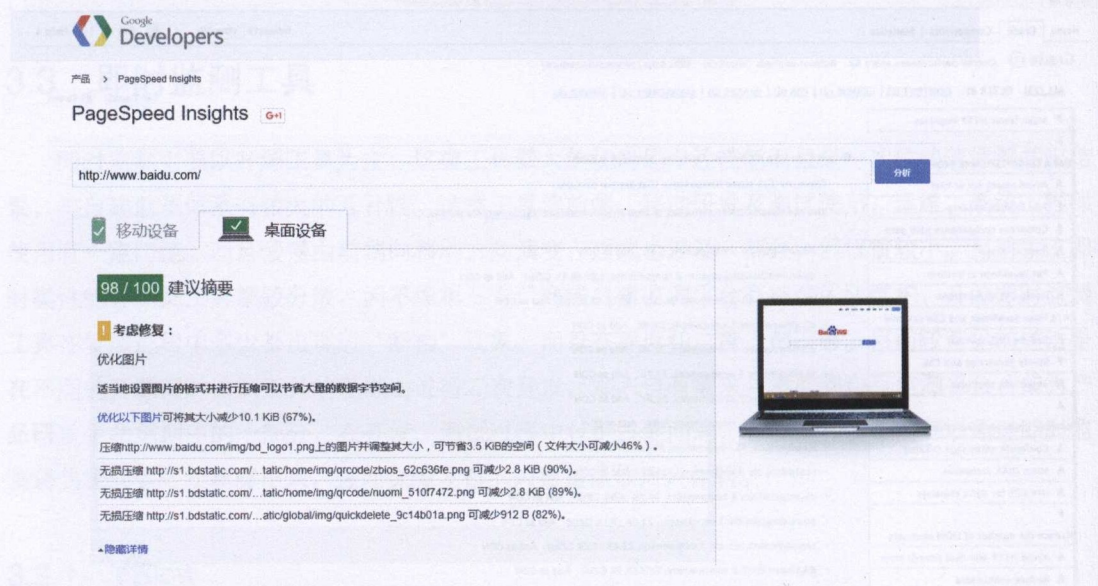


图3-4 PageSpeed Insights PC分析

3.3.3 WebPageTest

WebPageTest是Google开源项目“make the web faster”的子项目(“make the web faster包括page speed、spdy、tcpm等等”),它支持本地部署,同时也提供在线测试服务。WebPageTest通过调用PC或移动终端上真实的浏览器(IE、Firefox、Chrome、Safari等),模拟用户访问网页,对网页进行性能测试,并提供详细的分析和优化建议,WebPageTest界面如图3-5所示,基本功能如下。

- 1 支持多地域、多终端、多浏览器测试。WebPageTest在全球各地都部署了测试终端,包括美洲、亚洲、欧洲和非洲等地的主要地区都部署有终端,这些终端包括了PC和移动,覆盖IE、Chrome、Firefox等主流PC浏览器,也包括了Android和iOS的多种设备类型。
- 2 支持高级评测配置,如自定义UA、设置HTTP权限认证、自定义脚本执行等。自定义UA允许用户将WebPageTest伪装成任何浏览器或者将在UA中加入自己的特定标记,从而将测试流量

从正常流量中区别出来。设置HTTP权限认证允许用户对那些需要HTTP认证才能访问的页面进行测试。自定义脚本允许用户对页面执行复杂的事物操作，在执行操作的同时记录性能数据。

- 3 关键性能指标分析，如开始渲染时间、页面加载时间、整页时间等。在测试的过程中，WebPageTest会记录各个关键浏览器事件发生的时间，将这些时间作为关键指标展示出来，同时WebPageTest使用自己特有的算法判定网页的视觉加载进度和完全加载时间。
- 4 网页结构分析，包括资源组成、域名分析等。从资源角度，WebPageTest对页面内容进行分类统计，使用户可以一眼看出页面结构中哪些类型的资源占比最大，为性能优化找到关键目标。从域名角度，WebPageTest分别统计了页面中每个域名下有多少资源，以及每个资源的详细加载信息，便于用户从域名角度分析性能问题。
- 5 对每一个请求以及全局的性能分析、评分和建议。对于每一个请求，WebPageTest都会从是否开启Keep-Alive、是否开启Gzip、是否进行了图片优化、是否设置了合适的缓存、是否使用了CDN等多个角度进行评价，同时还对性能优化情况进行打分。

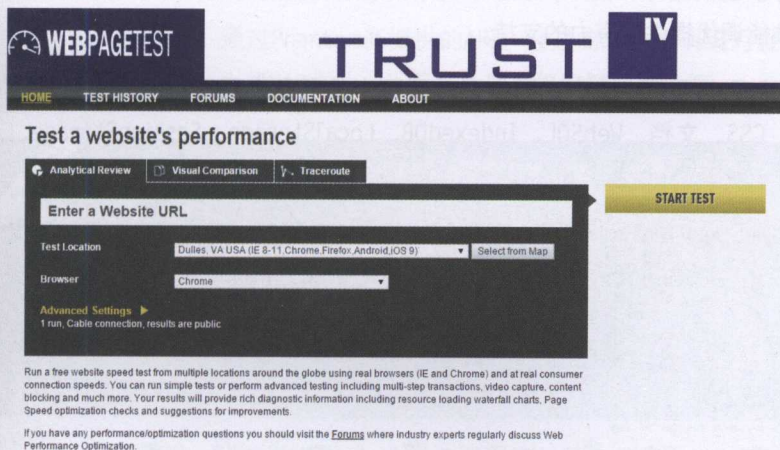


图3-5 WebPageTest官网

3.3.4 ChromeDevTools

ChromeDevTools是一套内置于谷歌Chrome浏览器，帮助Web开发人员检查代码面板，作为Web前端开发性能调试的必备工具。除了我们熟知的更改页面元素的标签代码就能看到页面效果外，还继承了很多高级的功能，只是应用于Chrome开发者工具的插件不同于普通的Chrome应用或者插件，它是给Chrome DevTools扩展更多的功能，方便查看和调试Web程序，为很多开发调试工作人员带来了

很多方便。另外，Chrome Dev Tool还允许开发者为其开发插件，例如Pagespeed和AngularJS等工具都有对应的插件，能够帮助开发者更加高效地使用对应工具进行应用开发。ChromeDevTools界面如图3-5所示，基本功能如下。

- 1 DOM检查功能。允许用户动态检查网页的DOM结构，对嵌套的DOM树进行展开折叠等操作。在网页中选中内容，在开发者工具中就能自动选中对应的DOM节点。在选中DOM节点的同时，用户能够编辑该节点的CSS，包括添加、删除和修改等操作。
- 2 命令行控制台。在控制台功能中，用户可以查看网页中使用console命令输出的信息，这给应用调试和代码跟踪提供了极大的方便。同时，用户还可以直接在控制台中执行JS代码，在网页加载完毕后手动调用网页中定义的函数或者访问网页中的变量等。
- 3 源码查看。在该功能中，用户能够按照域名分组查看网页加载的所有资源内容，静态对代码进行检查。对于JS代码，还能在该功能中对代码设置断点，进行单步调试和跟踪，设置跟踪变量，查看调用函数调用栈，查看事件侦听等。
- 4 网络瀑布图。网络瀑布图为用户详细记录了网页所发出的每一个请求，并且默认按照发起的先后顺序进行排序。在网络瀑布图中，展示了每个请求所花费的时间和每个请求的头信息，为网页调试和性能调优提供强有力的支持。
- 5 资源管理。在资源管理功能中，按照资源类型对网页中的所有资源进行了分类管理，资源包括图片、JS、CSS、文档、WebSQL、IndexedDB、LocalStorage、SessionStorage、Cookies、Cache和Service Workers等。
- 6 设备仿真。通过Chrome Dev Tool，用户可以将Chrome仿真成为移动设备，除了市场上常见的手机机型外，用户还可以自定义设备类型，指定设备尺寸和设备的UA，为移动开发调试提供了大力支持。

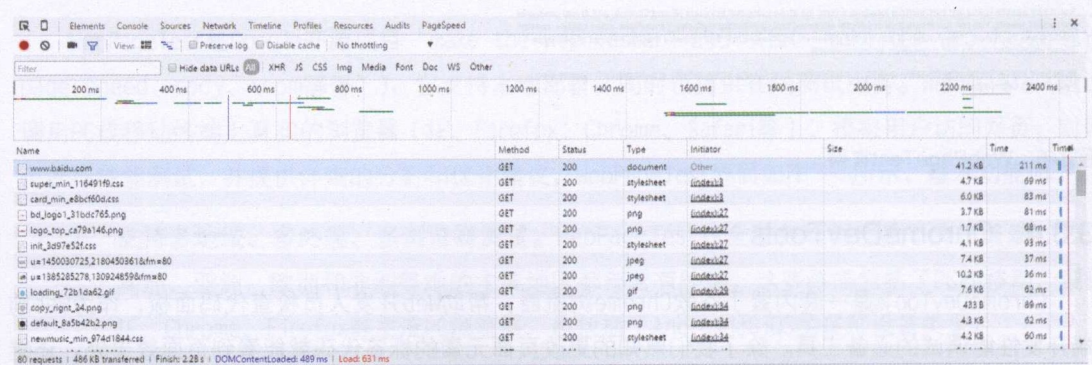


图3-6 ChromeDevTools网络分析

3.3.5 PhantomJS

PhantomJS基于JS的webkit内核无界面浏览器。这样访问网页就省去了浏览器的界面绘制所消耗的系统资源,比较适合用于网络测试等应用。它全面支持Web而无须浏览器支持,其快速,原生支持多种Web标准,主要使用场景是页面访问自动化、网络监控、无须浏览器的Web测试。功能很强大,使用也比较方便。但是它的“渲染”核心是QtWebkit,JS引擎是JavaScriptCore, PhantomJS工作在服务端,体积小速度快,支持各种浏览器特性,包括DOM操作、CSS选择器、JSON解析、Canvas绘图和SVG绘图等。PhantomJS界面如图3-7所示,基本功能如下:

- 1 无界面网站测试。和Jasmine、Qunit、Mocha等测试框架配合,进行应用的自动化功能测试。
- 2 页面截图。通过编写脚本进行网页内容的自动化截图工作,包括对SVG和Canvas内容的截图。除此之外,还能创建整张页面的截图和缩略图。
- 3 页面自动化操作。通过标准DOM API或者jQuery等工具,可以在PhantomJS中对网页进行自动化操作,包括点击按钮、滚动窗口、输入信息等。在此基础上,可以编写网页操作脚本,对应用进行用户操作级别的测试。
- 4 网络监测。通过PhantomJS提供的API,经过简单的代码编写就能够获得页面加载过程中的性能数据,生成HAR文件,为后续处理分析问题提供方便。

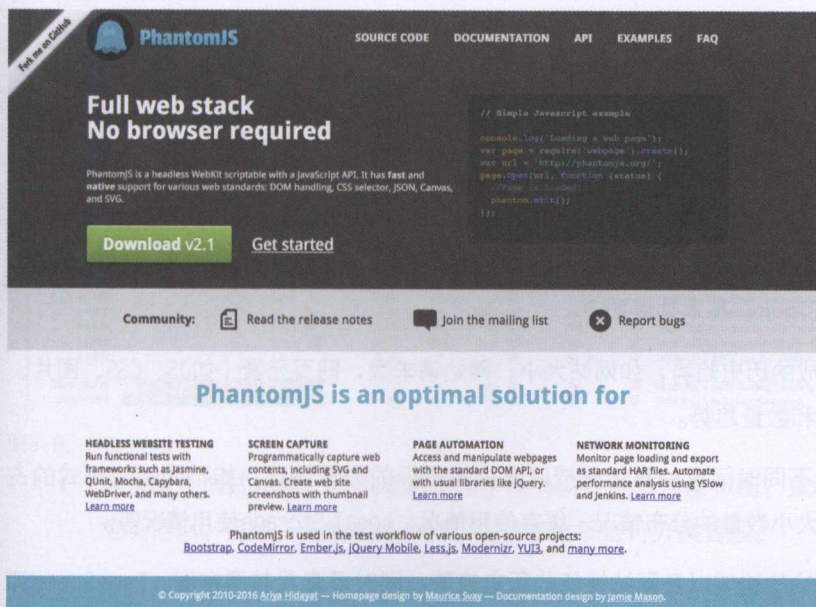


图3-7 PhantomJS官网

3.3.6 Jsp perf

Jsp perf是一个前端解决方案对比和性能比较的工具，对浏览器特性进行有针对性的性能调优，比较不同代码的执行效率。在前端开发中，解决一个问题常常有多种方案可以实施，然而可以通过Jsp perf来验证那种方案是最佳的，基本功能如下：

- 1 创建、运行、维护和分享JavaScript测试用例，从而为多人开发、技术社区讨论、开源合作等多种技术活动提供数据支持和测试方案。
- 2 同时比较多个不同JavaScript代码片段的运行性能，使用A/B测试的方式为同一个问题的不同解决方案提供性能评估支持，每个方案均会进行足够多次的测试，综合多次测试结果后再给出最终结果。
- 3 比较JavaScript代码在不同浏览器下的运行性能，支持包括IE、Chrome、Firefox在内的绝大部分浏览器，可以为用户提供代码在不同浏览器下运行的性能数据，使用户能够针对特定的浏览器进行专门的性能优化。
- 4 测试用例支持异步JavaScript的性能测试。前端开发中会有大量的异步逻辑，例如Ajax请求、事件处理等，JsPerf为这类逻辑提供了Async类型的测试，帮助用户发现异步代码中的性能问题。

3.4 其他工具

HTTP Archive

HTTP Archive由Steve Souders创建，通过对Alexa Top 1,000,000的网站进行长期的跟踪测试，对网站的构成和性能进行分析和记录，为探究Web技术的发展趋势、Web性能的研究提供数据支持。HTTP Archive界面如图3-8所示，基本功能如下：

- 1 查看全球网站构成的历史趋势，如网站大小，网站请求数，网页元素（如JS、CSS、图片、Flash等）的大小和数量趋势。
- 2 查看全球网站在不同时间点的具体构成以及关键指标的分布情况，如不同图片格式的占比、网页及元素大小数量的分布情况、缓存使用情况、Local Storage使用情况等。
- 3 查看某个具体网站的构成以及网站性能的历史趋势，同时可查看其瀑布图、PageSpeed评分、WebPageTest测试结果等性能数据。

HTML Transfer Size & HTML Requests

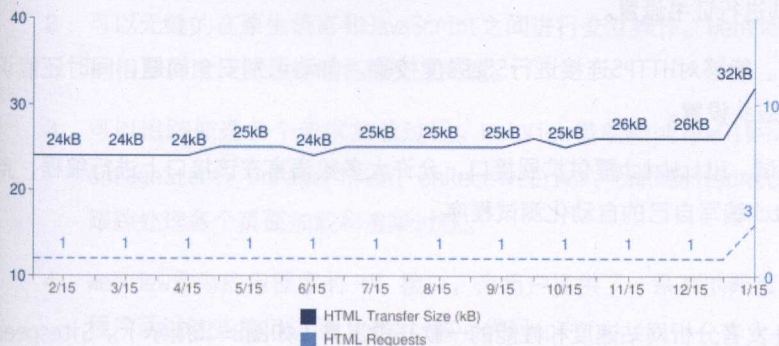


图3-8 HTTP Archive HTML分析

HttpWatch

HttpWatch是一款Windows下的抓包工具，可以和IE以及Firefox集成，抓取浏览器发出的HTTP和HTTPS请求，能详细展示每个请求的头信息、Cookie信息以及Payload信息等。此外，HttpWatch还提供iOS版本，支持iOS8以上的操作系统，同时支持iPhone、iPad和iPod等设备。HttpWatch在PC平台和移动平台上都分为基础版和专业版两个版本，其中基础版提供的功能有限，可以免费使用，专业版需要付费使用，提供的功能更加全面。HttpWatch界面如图3-9所示，基本功能如下：

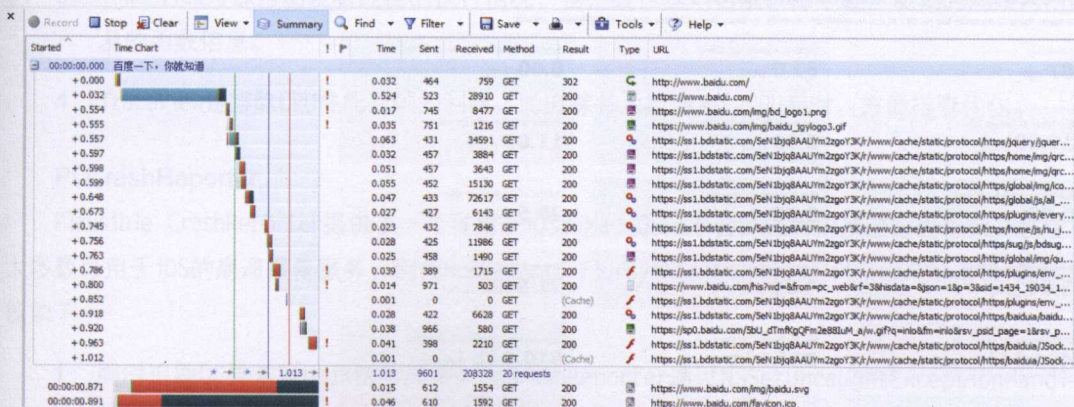


图3-9 HttpWatch

- 1 提供网页调试功能，可以查看页面中所有网页的请求瀑布图，查看每个元素的请求头和响应头信息。另外，还能对多个网页的请求过程进行分类管理。
- 2 提供网页性能调优功能，实时记录网页元素加载状况，毫秒级记录请求的每个时间阶段。能够记录网页应用中每个事件发生的时间，同时自动探测页面性能问题。

- 3 无须进行代理设置。较之传统的抓包工具，HttpWatch无须进行代理设置，安装简便，对于HTTPS的流量也无须进行证书设置。
- 4 提供安全测试功能。能够对HTTPS连接进行SSL强度校验，自动识别安全问题，同时还能识别不安全的请求响应头设置。
- 5 支持自动化HTTP测试。HttpWatch提供扩展接口，允许大多数语言在该接口上进行编程，用户可以利用HttpWatch编写自己的自动化测试程序。

Sitespeed

Sitespeed是一款帮助开发者分析网站速度和性能的一款开源工具（如图3-10所示）。Sitespeed通过分析网站的多个页面，基于最佳实践和指标规则，生成网站的性能分析报告，Sitespeed基本功能如下：

- 1 根据一定的规则遍历网站的页面，分析网站的整体性能情况分布以及每个页面的构成和性能详情。
- 2 提供Grunt插件，或通过TAP/JUnit XML输出与Jenkins等服务集成，实现自动化性能测试。
- 3 与Graphite、Grafana集成，提供快速部署自定义持续监测的解决方案。

36 pages analyzed for <http://www.cybercom.com>

Test performed Mon Dec 15 2014 22:16:30 GMT+0100 (CET) with sitespeed.io-desktop rules.

User-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 10_9_4) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/37.0.2062.120 Safari/537.36 Viewport: 1280x800



图3-10 Sitespeed分析视图

WebView

WebView是基于webkit引擎展现Web页面的控件。WebView功能强大，除了具有一般View的属性和设置外，还可以对URL请求、页面加载、渲染、页面交互进行强大的处理。WebView主要功能有：

- 1 加载渲染远程和本地的HTML数据。
- 2 可以无缝的在原生语言和JavaScript之间进行交互操作。WebView通过原生语言和JavaScript的相互调用，灵活处理各种交互问题。
- 3 可以追踪处理各个请求加载过程。WebView借助WebViewClient，通过onLoadResource、onPageStart、onPageFinish、onReceiveError、onReceivedHttpAuthRequest等回调方法，跟踪处理各个页面加载和渲染过程。
- 4 WebView作为应用程序的 UI 接口，为用户提供了一系列的网页浏览、用户交互接口，客户程序通过这些接口访问 WebKit 核心代码。

TraceView

TraceView是Android的一个可视化的调试工具，是Android平台特有的数据采集和分析工具，它主要用于分析Android中应用程序的性能问题。它可以通过图形界面的方式展示和跟踪程序的性能，并且能具体到method。TraceView基本功能：

- 1 TraceView能够查看跟踪代码的执行时间，分析哪些是耗时操作。
- 2 TraceView能够帮助跟踪函数的调用关系，尤其是Android Framework层的方法调用关系。
- 3 TraceView可以帮助查看线程的执行情况，包括线程基本信息、每个线程测试时间段内所涉及的函数信息。
- 4 TraceView是排除CPU性能瓶颈的利器，能追踪各函数调用的CPU耗时，方便排查优化。

PLCrashReporter

Plausible CrashReporter提供了一个可用于iOS和Mac OS X平台的进程内崩溃报告框架，包含有大多数可用于iOS的崩溃报告服务，例如HockeyApp, Flurry和Critictercism。PLCrashReporter基本功能如下：

- 1 同时追踪OC异常及UNIX信号异常。PLCrashReporter通过NSSetUncaughtExceptionHandler捕获OC层异常，通过捕获SIGABRT、SIGBUS、SIGFPE、SIGILL、SIGSEGV、SIGTRAP信号追踪UNIX异常。
- 2 提供完整的崩溃线程。PLCrashReporter提供应用崩溃时所有线程的方法调用堆栈信息，并且能帮助追踪崩溃问题到代码行。
- 3 提供开放API，易于集成到现有或自定义的Crash Reporter服务。

3.5 应用性能指标

应用性能关键指标用来衡量应用速度的快慢或体现网页访问过程中，系统、网络、前端、客户端等各环节渲染的快慢及问题，这里说的指标包括互联网PC端、移动互联网手机端及后端。通常横向看一个行业同一类型产品的速度快慢的时候，会使用几个核心指标来衡量，从中可以对比出与竞争对手在系统、网络、前端的差距，比如网络媒体行业，我们会对比腾讯、新浪、搜狐的门户、微博等，搜索行业会对比百度、Google、360的搜索等，不同维度的对比使用对应的指标来衡量，例如前端、后端、移动端等衡量的指标都不同，对应用性能的关键指标的理解对我们全国进行应用性能管理有非常大的帮助。通过了解这些指标项背后的含义、采集的方法、数值的高低对于真实用户体验的关联，以便诊断、优化页面和应用的性能问题。

3.5.1 用户指标

3.5.1.1 可用率

可用率也叫可用性，即业务或系统正常运行时间的百分比，是每个运维团队最主要的 KPI，也是 SLA（服务质量保证）中的一个重要度量指标，为运维SLA的实施和改进计划提供数据支持。通常将可用性分为业务可用性、系统可用性两类。可用率计算公式为(运维服务时间- Σ 维护影响时间- Σ 故障影响时间)/运维服务时间*100%，运维服务时间是以每天24小时计算，每周为7*24小时，以此类推。故障影响时间为故障或维护事件的发生时间至结束时间。可用率衡量范围是非常广泛的，可以适用任意应用性能，例如产品、系统、网页、元素、功能模块、IDC、CDN等，无论是PC、移动主流端到端性能监测，还是更底层的应用、系统、网络性能监测都需要可用率作为最核心的指标，通过可用率可以直接衡量这些监测对象为用户提供服务的质量状况，可用率与应用性能有直接关联，差的应用性能将导致低可用率，通常有以下几类衡量标准，如表3-11所示。

表3-11 可用率衡量标准

描述	通俗叫法	可用率级别	年度停机时间
基本可用率	2个9	99%	87.6小时
较高可用率	3个9	99.90%	8.8小时
具有故障自动恢复能力的可用率	4个9	99.99%	53分钟
高可用率	5个9	100.00%	5分钟

3.5.1.2 事务

应用性能中的事务主要指Web事务，即用户某一步或几步操作的集合，比如用户对某一个页面的一次请求，用户对某系统的一次登录，用户对商品的一次确认支付过程，这些都可以看作一个事务，事务有一个开头和一个结尾，它们指定了事务的边界，往往Web事务是后端逻辑并由语言逻

辑、数据库的交互决定性能。事务用来衡量代码中一行代码或多行代码的执行所耗费的时间。可以将事务开始放置在脚本中某行或者多行代码的前面，将事务结束放置在该行或者多行代码的后面，在该脚本的虚拟用户运行时，这个事务将衡量该行或者多行代码的执行花费了多长时间。

互联网产品是具有商业价值的功能集合，每一个产品功能又需要由一个或多个Web事务完成，事务可以理解为是产品功能的最小单元，也可以理解为是产品应用性能的最小载体，需要找出影响用户体验的事务，进而改进产品体验。对于一个上线运营在生产环境的互联网产品，每天都有多个地域的用户使用不同的终端访问会产生大量的事务操作，这些事务操作主要决定了产品的后端性能，通过事务监测可以将后端性能完全透明化。

3.5.1.3 吞吐量

吞吐量是指系统在单位时间内处理请求的数量。一个应用的吞吐量（承压能力）与request对CPU的消耗、外部接口、IO等紧密关联。单个request对CPU消耗越高，外部系统接口、IO影响速度越慢，系统吞吐能力越低，反之越高。对于面向用户的Web服务，响应时间（或者系统响应时间和应用延迟时间）可以很好地度量系统的性能，但对于并发应用模块或系统，通常需要用吞吐量作为性能指标。在处理少量请求时，在每个时间点都可能有许多资源被闲置，当处理多个请求时，如果资源配置合理，每个用户访问平均响应时间并不随用户数的增加而线性增加。实际上，不同系统的平均响应时间随用户数增加而增长而变化的，这也是采用吞吐量来度量并发系统的性能的主要原因。一般而言，两个具有不同用户数和用户使用模式的系统，如果其最大吞吐量基本一致，则可以判断两个系统的处理能力基本一致。

吞吐量是一个比较通用的性能指标，应用性能问题发生与吞吐量超载有直接的关系，往往超载与否的判断也可以通过应用性能变化作为依据。一个互联网企业或产品最理想状态是在去除应用性能问题并保持高负载、高可用，将性能与运营成本最大化，因为性能与成本是伴随产品生命周期变化而变化，往往要达到绝对平衡是较困难的，需要足够的经验和体系的监测平台。吞吐量示意图如图3-11所示。

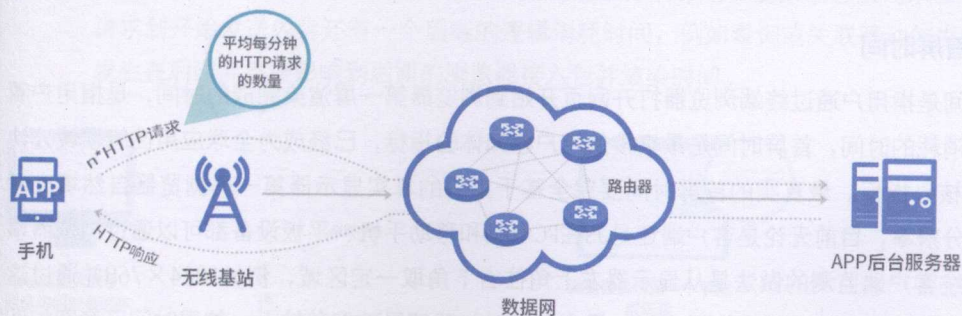


图3-11 吞吐量拓扑图

3.5.1.4 响应时间

响应时间是指系统对请求作出响应的时间。这个指标与用户使用产品的主观感受是非常一致的，因为它完整地记录了应用系统处理请求的时间。由于一个产品通常会提供许多功能，而不同功能的处理逻辑也千差万别，因而不同功能的响应时间也不尽相同，甚至同一功能在不同输入数据的情况下响应时间也不相同。所以在讨论一个系统的响应时间时，通常是指该系统所有功能的平均时间。对于应用模块或API的响应时间小于100ms应该是不错的，响应时间在1s左右可能属于勉强可以接受，如果响应时间达到3s就完全难以接受了。可以把用户感受到的响应时间划分为“浏览器内容渲染时间”（前端）和“后端系统响应时间”（后端），前者是指客户端的浏览器在接收到网站数据时呈现页面所需的时间，而后者是指服务端接收到用户请求到浏览器接收到服务器发来的数据所需的时间。显然“浏览器内容渲染时间”与用户本地计算机、浏览器、网络有关，“后端系统响应时间”与应用的逻辑和功能有关。

3.5.1.5 同时并发数

同时并发数或叫并发用户数，是指系统可以同时承载的正常使用系统功能的用户的数量。与吞吐量相比，并发用户数是一个更直观但也更笼统的性能指标。实际上，并发用户数是一个非常不准确的指标，因为用户不同的使用模式会导致不同用户在单位时间发出不同数量的请求。以Web应用为例，假设用户只有注册后才能使用，但注册用户并不是每时每刻都在使用该网站，因此具体一个时刻只有部分注册用户同时在线，在线用户就在浏览网站时会花很多时间阅读网站上的信息，因而具体一个时刻只有部分在线用户同时向系统发出请求。这样，对于Web应用我们会有三个关于用户数的统计数字：注册用户数、在线用户数和同时发请求用户数。由于注册用户可能长时间不登录网站，使用注册用户数作为性能指标会造成很大的误差。而在线用户数和同时发请求用户数都可以作为性能指标。相比而言，以在线用户作为性能指标更直观些，而以同时发请求用户数作为性能指标更准确些。

3.5.1.6 首屏时间

首屏时间是指用户通过终端浏览器打开网页开始到浏览器第一屏渲染完成的时间，是指用户看到第一屏所消耗的时间，首屏时间是最直接的用户感知体验指标，已经成为全球应用性能领域公认的最重要的核心指标。最真实的首屏时间要完全基于用户的真实显示器第一屏浏览器自然填充并自适应所有分辨率，目前无论是客户端还是JS在PC电脑和移动手机、平板设备都可以通过浏览器事件得到。传统客户端监测的做法是从显示器左上角往右下角取一定区域，例如1024×768并通过这区域中的12个点读屏取第一屏，轮播广告、异步和延时加载都导致偏差较大。首屏时间示意图如图3-12所示。

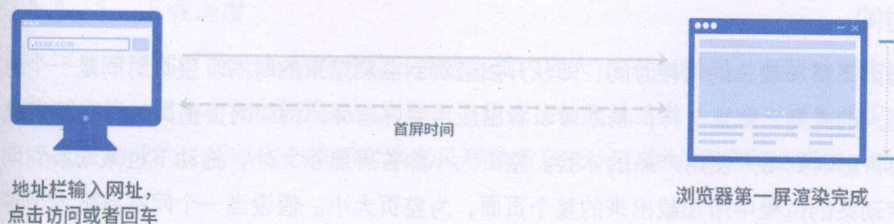


图3-12 首屏时间拓扑图

3.5.1.7 白屏时间

白屏时间是指用户输入URL确认后, 由浏览器发送HTTP请求结束开始, 到浏览器接收到服务端发送的数据包并开始填充浏览器第一屏的时间, 即浏览器开始显示内容的时间, 白屏时间是仅次于首屏时间的用户感知体验指标。白屏时间在慢网络环境(宽带环境)下最为明显, 无论浏览器当前页面有没有旧网页, 当浏览器开始填充新网页时, 都会从空白内容开始渲染, 不断显示内容, 直到内容慢慢加载完。但是在传统的采集方式里, 是在HTML的head标签结尾里记录时间戳, 来计算白屏时间。在这个时刻, 浏览器开始解析body标签内的内容。而现代浏览器不会等待CSS树(所有CSS文件下载和解析完成)和DOM树(整个body标签解析完成)构建完成才开始绘制, 而是马上开始显示中间结果。所以经常在低网速的环境中, 观察到页面由上至下缓慢显示完, 或者先显示文本内容后再重绘成带有格式的页面内容。在更深入的优化工作中, 会将cookie内容和head标签的字符数也考虑进来。将这部分内容的数据量控制在少数几个TCP包的净载荷内, 在有限的网络带宽下, 赶在TCP的拥塞控制机制暂缓传输过程前就传输完, 从而以较小的延迟开始绘制页面。白屏时间示意图如图3-13所示, 通常影响白屏时间的因素有下面几个方面:

- 1 网络因素, 用户本地与应用服务器端之间的网络延时一定程度上决定了白屏时间长短, 特别是移动Web App应用。
- 2 后端因素, 通常简单的Web应用基本不受后端服务的影响, 复杂动态应用从服务端接入到请求到开始发送内容还有一个后端的逻辑消耗时间, 例如查询或关联等动作发现的消耗都发生在后端并直接影响到后面的浏览器接入包并渲染时间。

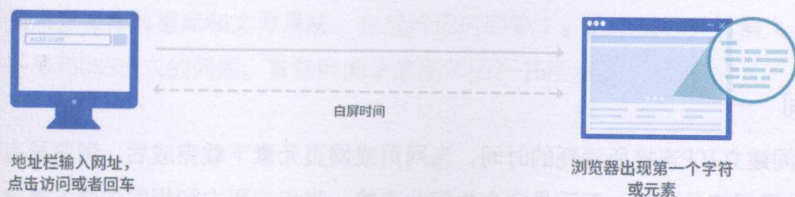


图3-13 白屏时间拓扑图

3.5.1.8 整页时间

整页时间即网页完整加载总的消耗时间，即从开始监测到监测结束的时间。整页时间是一个通用性能指标，如同人的身高、体重一样，基本可以看出应用的好与坏，例如网页整页时间在2~3s是较优秀的，超过5s将影响到用户使用产品的体验。整页大小除首屏显示之外，拖动下拉滚动条到网页底部，在拖动滚动条的过程中所加载出来的整个页面，为整页大小。假设当一个网站首页大小等于首屏大小，那么整页时间将会等于首屏时间，但是绝大多数网站在首屏加载完成后，还有很多内容未完成加载，若用户没有在整页加载完成之前选择关闭浏览器，那么这些内容会被持续加载，直到加载完成，从页面加载出第一个元素开始，到整个网页被加载完成的时间为整页时间。整页时间示意图如图3-14所示。

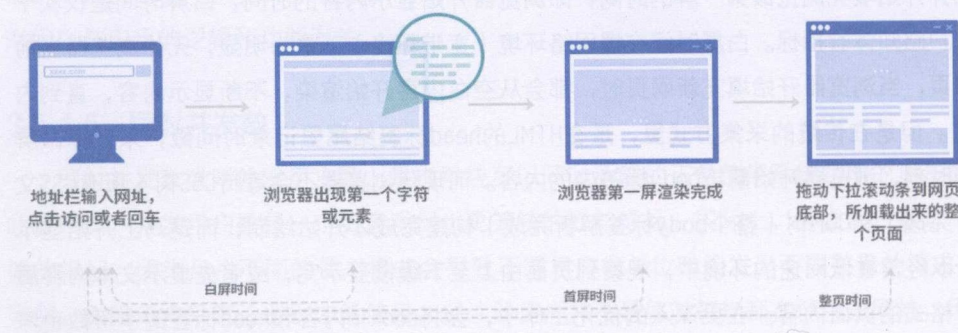


图3-14 整页时间拓扑图

3.5.1.9 DNS时间

DNS (Domain Name System) 即域名解析服务，DNS时间是从浏览器终端访问请求开始，到浏览器终端获得最终访问主机IP地址的时间。发生一次域名解析后，Windows操作系统会缓存此操作的结果，当监测节点再次解析相同的域名时，Windows会将此缓存的结果返回给监测节点，对应消耗时间值可能为0。DNS解析过程其实是极其简单的，但往往是影响应用的一个重要环境，例如新域名，全国各地localdns缓存较少的情况下，这个新域名下的所有网页元素都会受影响，启用新域名主要是新产品、新模块、上线第三方CDN、使用第三方DNS解析服务等这些不可抗的情况下发生，所以需要尽早规避。

3.5.1.10 建立连接时间

浏览器和Web服务器之间建立TCP连接所消耗的时间，当网页或网页元素下载完成后，浏览器有可能会根据服务器返回的结果保持此连接，而不是完全关闭此连接。当用户再次和相同的服务器建立连接时，会复用此连接，对应消耗时间可能为0。

3.5.1.11 下载速度

下载速度即页面大小/网络传输时间或下载内容大小/下载时间。下载速度主要用来衡量越大的文件越准确，因为小文件还没有达到最大下载速度就已经结束了，会存在失真现象，而网页是由多个小元素构成的，所以这个指标对网页只有参考意义。下载速度偏网络属性，主要用来衡量IDC、CDN的性能。

3.5.1.12 网络传输时间

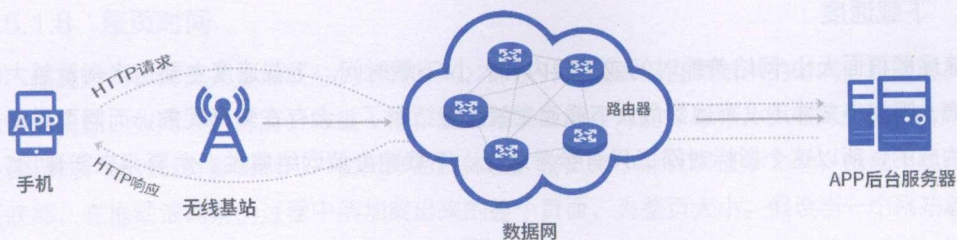
监测一个页面时，发生网络通讯的总消耗时间。网络传输时间约是DNS时间、建立连接时间、发出请求时间、网络首包时间、内容下载时间、关闭连接时间之和，与下载时间一样是网络属性较强的指标。

3.5.1.13 后端响应时间

浏览器发出HTTP请求后，Web服务器进行响应以及后台处理的时间。后端响应时间主要指生产环境上的应用性能，包含了后端所有的逻辑交互和外部调用时间，通常后端响应时间关联是最难可视化的，性能优化的难点也在于此，后端优化的代价最大，投入与收益往往不成正比。

3.5.1.14 首包时间

浏览器发送HTTP请求结束开始，到收到Web服务器返回的第一个数据包的消耗时间。此指标包含了发送HTTP请求时最后一个数据包在网络上的传输时间、服务器响应此请求的时间和服务器回应的第一个数据包在网络上面的传输时间。首包时间与白屏时间较接近，相比少了浏览器本地接入数据和开始渲染所消耗的时间（包括了浏览器内存初始分配，解析下载的数据包，页面显示等多个小段时间的总和），即没有发生网络通讯的时间片断的总和。如果从Navigation Timing API上采集，就是responseStart减去navigationStart。navigationStart这个数值，是指用户在浏览器地址栏输入URL和回车、或者点击页面上一个链接、或者按下了刷新页面按钮的时刻。在传统的采集方法中，也有在HTML的head标签内最顶部的位置使用JavaScript记录时间戳。首包时间的意义在于，它正好将浏览器的问题隔离开来，主要包括了网络时间（DNS解析、建立TCP连接等）和服务器时间（处理请求、操作数据库和文件系统、生成响应内容等）。这个指标突然偏大，可以用来快速定位后端服务器和DNS请求的问题。首包时间示意图如图3-15所示。



从APP发出HTTP请求开始，收到服务器返回的第一个数据包所消耗的时间

图3-15 首包时间拓扑图

3.5.1.15 基础页时间

基础页面即Web服务器返回的纯文本HTML文件，基础页面下载字节数即为该纯文本HTML文件的总字节数。因为基础页是浏览器加载的第一个请求，进而导致基础页时间直接影响首屏时间，加上基础页是唯一的决定所有网页渲染过程的，所以需要做得足够轻，足够简洁。因为基础页决定了网页渲染过程，无论是前端还是后端JS都是添加到基础页中，甚至是Server端自动内容和网络优化，也是在基础页改。基础页时间示意图如图3-16所示。

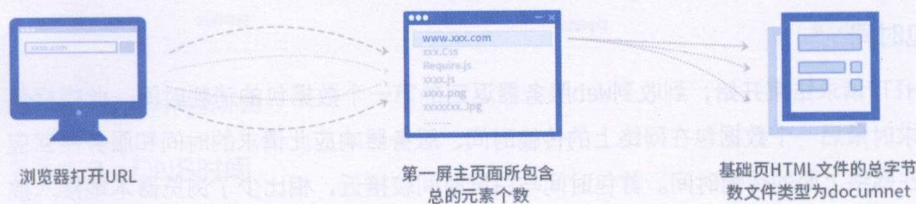


图3-16 基础页时间拓扑图

3.5.1.16 页面大小

浏览一个页面过程中从服务器返回的网络通讯字节总数。此指标包含了HTTP协议头的字节数，代表了实际发生的真实的网络流量，这个指标与总下载时间一样，好比人的身高和体重，是一个通用的指标，能过页面大小可以看出网页的胖瘦，进而推测相关性能问题。此指标当服务器启用HTTP压缩时，页面内全部元素总下载字节数的累计会比页面总下载字节数稍微小。而在Web服务器返回的网络数据包是经过HTTP压缩的情况下，浏览器会对原始网络数据包解压缩，此指标指的是解压缩后的网页大小，页面内全部元素总下载字节数的累计会比页面总下载字节数大。页面大小示意图如图3-17所示。



图3-17 页面大小拓扑图

3.5.2 服务器指标

3.5.2.1 CPU使用率

CPU使用率即主机中运行的程序占用的CPU资源的比例，越大代表系统越繁忙，日常平均总使用率保持在75%以上较理想。

3.5.2.2 平均负载

平均负载 (load average) 可以理解为每秒钟CPU等待运行的进程个数，它所包含的信息是在一段时间内 CPU正在处理以及等待 CPU处理的进程数之和的统计信息，也就是 CPU使用队列的长度的统计信息，每核保持在1~3最好。

3.5.2.3 内存使用量

内存使用量指系统、用户和进程使用内存所消耗的内存大小，内存资源的充足与否直接影响应用系统的使用性能。直接从物理内存读写数据要比从硬盘读写数据要快得多，而物理内存是有限的，这样就引出了物理内存与虚拟内存的概念。物理内存就是系统硬件提供的内存大小，是真正的内存，相对于物理内存，在Linux下还有一个虚拟内存的概念，虚拟内存就是为了满足物理内存的不足而提出的策略，它是利用磁盘空间虚拟出的一块逻辑内存，用作虚拟内存的磁盘空间被称为交换空间。一般来说如果空闲内存/物理内存大于70%，内存性能优；如果小于20%，则性能差。

3.5.2.4 IO使用率

磁盘IO使用率分为磁盘读IO和写IO，及当前读写IO占整个磁盘IO的比例。

3.5.2.5 IO读写速度

IO读写速度即每秒进行读写 (I/O) 操作的大小，也称吞吐量，主要体现硬盘传输数据流的速度，传输数据为读出数据和写入数据的和。其单位一般为 Kb/s, MB/s等，当传输大块不连续数据的

数据，该指标有重要参考作用。

3.5.2.6 磁盘操作数

磁盘操作数（IOPS，Input/Output Per Second），即每秒进行读写（I/O）操作的次数，即每秒的输入输出量（或读写次数），是衡量磁盘性能的主要指标之一。IOPS是指单位时间内系统能处理的I/O请求数量，一般以每秒处理的I/O请求数量为单位，I/O请求通常为读或写数据操作请求。固态硬盘SSD是一种电子装置，避免了传统磁盘在寻道和旋转上的时间花费，存储单元寻址开销大大降低，因此IOPS可以非常高，能够达到数万甚至数十万。

3.5.2.7 网络带宽

网络带宽即每秒发送和接收的数据的大小，通过网络带宽基本能判断系统和应用负载情况，非常适合静态应用。除此以外，服务器上的网络指标还有延迟、阻塞、冲突、丢包、吞吐量、速率等，其中延时与丢包与性能关联最紧密。

- 1 网络延迟是指在传输介质中传输所用的时间，即从报文开始进入网络到它开始离开网络之间的时间，延迟值越低速度越快。延迟与服务器所在的地域和运营商有直接关系，覆盖范围越远，延时越大。
- 2 网络丢包是指数据在网络传输过程中数据包丢失，每个数据包中有表示数据信息和提供数据路由的帧，而数据包在网络介质中传播因物理线路故障、设备故障、路由信息错误等原因导致有一小部分丢失。

3.5.2.8 连接数

连接数指系统当前网络链接状态，这个指标主要体现真实用户与服务器间建立的交互。连接数示意图如图3-18所示。

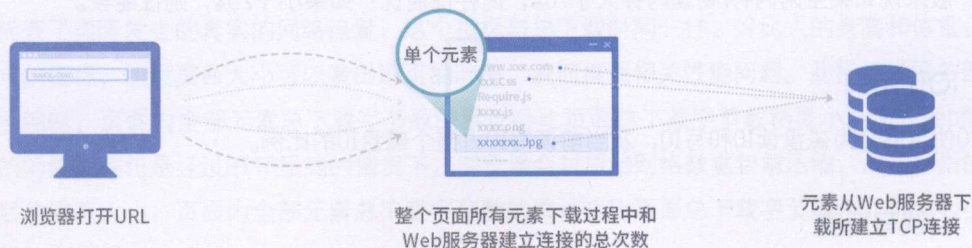


图3-18 连接数拓扑图

3.5.3 移动指标

3.5.3.1 启动时间

启动时间是指APP启动过程消耗的时间。

3.5.3.2 响应时间

响应时间是指发送 HTTP 请求开始，到收到所有响应内容的时间。

3.5.3.3 吞吐量

吞吐量是指平均每分钟的 HTTP 请求数量，单位为：rpm (Requests per minute)。

3.5.3.4 崩溃率

崩溃率是指出现崩溃事件的数量和应用启动次数（会话数）的比率。

3.5.3.5 活跃会话数

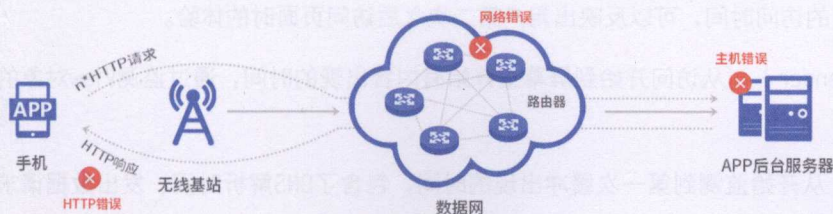
活跃会话数是指用户交互过程中每个 App 终端每分钟与服务器的连接次数。

3.5.3.6 传输数据量

传输数据量是指总的数据传输量。单位为Byte或KBytes。

3.5.3.7 错误率

错误率是指HTTP错误数量与请求数量的比率。错误率示意图如图3-19所示。



在监测时间段内，发生的总的错误数（包括HTTP错误和网络错误、主机错误）/总的请求数

图3-19 错误率拓扑图

3.5.4 其他指标

TPS每秒事务数

TPS (Transaction Per Second)每秒钟系统能够处理的交易或事务的数量，它是衡量系统处理能力的重要指标。

并发

并发分为狭义和广义两类。

- 1 狭义的并发，即所有的用户在同一时刻做同一件事情或操作，这种操作一般针对同一类型的业务或者所有用户进行完全一样的操作，目的是测试数据库和程序对并发操作的处理。
- 2 广义的并发，即多个用户对系统发出了请求或者进行了操作，但是这些请求或操作可以是不同的。对整个系统而言，仍然有很多用户同时进行操作。

狭义并发强调对系统的请求操作是完全相同的，多适用于性能测试、负载测试、压力测试、稳定性测试场景，广义并发不限制对系统的请求操作，多适用于混合场景、稳定性测试场景。

响应时间

响应时间是指从客户端发一个请求开始计时，到客户端接收到从服务器端返回的响应结果结束所经历的时间，响应时间由请求发送时间、网络传输时间和服务器处理时间三部分组成。

首次访问 (FV=First View)

不带任何缓存情况下页面的访问情况，可以反映出用户首次访问页面时的体验，目前PC真机监测主要是这种类型。

重复访问 (RV=Repeat View)

带缓存情况下页面的访问时间，可以反映出用户第二次之后访问页面时的体验。

开始渲染时间 (Render)：从访问开始到屏幕上开始有内容出现的时间，通过监测View对象的变化得到。

缓冲前准备时间：从开始监测到第一次缓冲出现的时间，包含了DNS解析时间、发出数据请求及接收第一个数据包的时间，这个指标是流媒体监测的重要指标。

TCP连接时间：浏览器和Web服务器建立TCP/IP连接的消耗时间。当元素下载完成后，浏览器可能会根据服务器返回的结果保持此连接，而不是完全关闭此连接。当监测节点再次和相同的服务器

建立连接时，会复用此连接，对应消耗时间可能为0。

缓冲时间：流媒体播放器处于缓冲状态持续的总时间。

等待时间

等待时间 = 连接时间 + 首次缓冲时间 + 所有再缓冲时间，等待时间是一个重要的指标，系统用此值来表示流媒体文件监测的性能。

首次播放时间：

流媒体播放器开始播放的时间，如果在一次监测中出现了再次缓冲，则此时间指的是开始播放到出现再次缓冲所花费的时间。

用户可操作时间：

用户可以进行正常的点击、输入等操作。

自定义的时间点：

对于研发来说，需要自定义一些个性化的性能数据，例如：某个组件init完成的时间、某个重要模块加载的时间等。

第4章 性能监测平台搭建实践

4.1 为什么要搭建监测平台

对用户体验和应用性能监测相信任何一家互联网公司都不会怠慢，特别是互联网企业都有基本共识，好比自然灾害，如果在灾害发现前监测到并采取措施规避，会将损失和伤亡缩小数倍。当一个产品初步进入规模和稳定期，无论现在与未来，都需要一套符合产品特性并行之有效的监测体系来保驾护航，而且这套监测平台是完全与产品特性结合在一起，并一起成长、演变才有意义。

第三方厂商有先天性短板

可以很肯定几乎所有第三方厂商提供的SaaS版和定制大客户版都不可能达到或帮助企业实现企业级的应用性能管理和优化，主要有以下几点短板。

- 1 数据安全性，最有价值的应用性能数据来自产品的真实用户和服务后端，而这些数据都是企业最宝贵的数据，如果通过第三方实现监测将直接使业务接口及用户暴露出去。
- 2 没有针对性，产品迭代过程是会发生巨大的变化，无论从产品形态还是架构，这些都需要时刻随产品变化调整监测，另外产品形态的多样性和特殊性与第三方监测通用性是矛盾的，第三方同时支持多个企业特性基本不可能。
- 3 沟通成本高，甲乙双方关系与公司内部员工之前的沟通效果会有天壤之别，特别是To B的第三方服务企业，同时要支持多家企业需求，从需求沟通到理解到实现，这个过程是漫长的。

自建监测平台可以满足多数人

企业搭建监测平台主要为第一时间发现、定位并修复性能问题，从而提高用户体验，实现这个过程中，需要满足产品相关的研发、测试、运维等多个项目内外部角色需求，只有参与项目中的各角色才最清楚各自负责的环节会发生或容易发生那些性能瓶颈。例如前端研发工程师最关心页面渲染过程，移动研发工程师最关心不同制式、手机厂商的用户使用APP的差异，运维工程师侧重系统、网络的性能。往往这些需求都有项目的特殊性，不同企业不同部门不同业务都是不一样的，只有自建平台才可以消化和针对性的落地实现。即使深度的业务自有的性能数据，项目各角色也容易在代码中自行进行采集，统一接入到公司级的监测平台展现，各角色的收益如下。

- 1 对于运维人员，通过对各层次的数据的展示和告警设置，快速直观的发现和定位故障。
- 2 对于研发、测试人员，通过对各层次的数据的展示，来反应业务的容量和性能，通过设置阈值来对业务的容量和性能管理。
- 3 对于公司中高层，通过对各维度、各层次数据的量化，来量化业务运行的状态。
- 4 对于所有人，用数据说话很容易定位故障、分清楚责任，即使遇见重大故障，都可以在监测数据中评估出影响范围和损失。

自建监测平台可以随产品变化

自建监测平台最大的益处在于可以伴随产品整个生命周期，从产品设计阶段就可以很清楚地知道有多少模块，需要部署在那些机房，需要多少服务器，数据流是怎么样的等，在产品研发和上线过程中就可以将监测集成进去，从而一开始就成为产品的一部分，伴随产品的不断迭代而一起发生变化，例如前端主要通过内嵌JS代码来实现真实用户的访问过程体验监测，新版本在开发过程中就可以直接内嵌监测代码，移动研发工程师也可以直接加载监测SDK。部分周期性针对新版本或特殊用户进行的监测也可以随版本的迭代随时上下架。

自建监测平台具有重要战略意义

自建监测平台的开发和维护方主要为运维团队或性能优化团队，这样的平台建立和定位从一开始就具有可持续保障产品用户体验的使命。不但统一公司在线服务用户体验指标定义和数据口径，更为公司管理层了解服务动态、工程师研发质量提供数据支持，而且建立全面的用户访问速度监控平台、提供统计和报告服务更助力重点产品线提升访问速度。可以看出应用性能监测平台对企业具有重要战略意义，从一开始就具有较强生命力，对负责研发和维护的团队也具有业绩支撑，国外优化互联网企业及国内BAT都投入了大量的人力和资源搭建自有监测平台，例如百度从2010年立项，用了4年时间建立了百度企业级的应用性能监测平台，不断吸收产品线需求及研发前沿应用性能监

测功能，协助百度14个核心产品持续进行性能优化。

4.2 如何搭建性能监测平台

对于互联网企业团队无论是研发、测试还是运维团队都在朝平台化和自动化努力，这也是互联网发展和演进的必然，伴随规模效应不断放大，量变到质变，通过增加人力已经不可能应对，平台化是实现一切可自动化事务的必然选择，已经成为我们的武器和未来，BAT都是往这个方向在进行中，结合个人在百度负责性能优化的经历，同时要为100多个产品服务，同时监测任务超过2000个，产品线需要和监测数量都不可能依赖第三方来实现，部分需求通过第三方实现也会发生大量持续的成本，只有通过自建性能监测平台来消化，而且自建性能监测平台可以通过API集成到其他部门的平台，可持续最大化平台价值，往往这些部门就是主要的受众和服务对象。

最大化受众并可持续

应用性能监测平台的作用相信所有人都容易理解，在用户至上的时代，企业上下所有人都愿意花时间和精力在改善用户体验上，这些愿意改善用户体验的人才主要受众，他们分布在不同的部门不同的产品组，从受众规模和部门看研发部>质量部>运维部>网络部，而这些受众部门和团队对应的需求也是不同的：

- 1 研发部，主要是移动开发工程师需要移动SDK监测，Web前端工程师需要JS监测，同时需要移动、PC真机监测了解竞品差异，目前只有真机监测才可以监测到竞品的性能。
- 2 质量部，因质量部主要对口研发部，需求与研发部类似，也是即时监测最大的受众，因为需要测试多个版本和支持快速迭代，快速全面的PC和移动即时监测是他们的首选。
- 3 运维部，运维部与所有部门交集最大，也是性能监测平台的发起方和建设方；需求范畴最广，从移动、PC终端用户监测到网络、系统、应用监测都需要，更深层次的开发语言、关联关系、语义、主流第三方平台等也是运维部要关注的范畴。
- 4 网络部，主要需求网络监测和PC真机监测，这两类监测都支持竞品监测，在评估IDC质量和稳定性，IDC覆盖性能等都有重要意义，也是网络交付最重要的量化来源，以上主要指外网。

自建为主，第三方作为辅助

严格意义上看，当前应用性能监测还没有开源的方案，只有两个途径实现，分别是自建和使用第三方厂商。自建，顾名思义是针对企业业务特性，定制和实现应用性能监测，进而达到量测和改善应用性能的目的，大的互联网公司肯定会选择自建，会有一个专职团队将性能监控、分析和优化

结合起来，产生一整套行之有效的电子流，将产品性能监测、分析和优化生产线化，不断持续改善产品用户体验。第三方厂商的角色和作用主要有以下几个方面。

- 1 覆盖面广，基本拿来就可以用，部分领域涉及较早，通常比较擅长，例如国内老牌应用性能监测厂商基调网络在端到端的PC真机监测做得较完整，可以与自建平台互补，但整体产品体系离个性化的企业需求差距较大。
- 2 发展迅速，2014年是国内外APM行业发展井喷的一年，第三方应用性能监测产品在资本市场及国外优秀企业上市推动下开始有质的飞跃，APM行业由此进入2.0时代，国内在过去多年都以PC真机监测作为主要营收来源升级为涵盖移动、应用、网络等众多应用性能产品体系，但因积累和发展时间较短，还未能真正具有商业价值，但在APM2.0时代企业与第三方厂商起步基本一样，都是同时发展，相信需要时间来验证优劣，或许在大量资本和人力投入下会产生卓越的APM企业并繁荣整体行业。
- 3 前期互补，小中企业或初创企业在人力和时间有限的情况下，借助第三方厂商来快速发现问题并加以改进，以时间换空间，而且部分SaaS免费服务是没有成本的，中大企业可以在跟随第三方厂商发展吸收有益的营养，结合企业自身产品特性，与时俱进。
- 4 仅发现问题，结合多年对第三方厂商使用和合作经验，第三方厂商因不能深入走进企业内部及整体产品生命周期，一直停留在浅层的“力所能及”的性能数据采集上，还远未到帮助企业深度分析问题及解决问题，更谈不上配合产品线做持续的大规模的性能优化。

持续不断完善才有竞争力

从前面的分析可以看出无论是自建还是第三方厂商都需要一个发展阶段，因为产品类型众多和业务都在变化发展，不存在快速立项建一套大而全的应用性能监测手段能覆盖所有产品线和业务，个人的体会当前的应用性能监测与传统的运维监控是完全不同的两回事，运维监控对象主要为基础资源，例如系统、网络、功能模块等，侧重监测指标及是否可用，而应用性能监测的重心是测试和度量，监测对象与真实用户感知及影响用户感知的前后端所有环节，侧重改善和优化用户体验，自建应用性能监测平台可分为以下三个阶段实现。

- 1 初期。搭建平台和建立优化方法论，监控系统和性能优化技术研究，性能分析、推广。以第三方厂商为主快速切入，将欲取之，必先与之，通过第三方厂商快速从无到有实现必要基础的性能监测，例如网站性能、网络性能等，辅助优化前端性能和网络性能、网络分布等，尽快修复大粒度的性能问题。
- 2 中期。以自建应用性能监测平台为主，第三方为辅助，重点实现基于真实用户的移动SDK

和移动Web APP监测、PC JS监测、网络监测、应用监测等，开始建立体系的产品性能监测与优化，覆盖核心产品线，监控系统和优化工具的完善和全面推广，核心产品优化方案定制化，建立核心产品性能趋势并长期跟踪。

- 3 后期。以建立企业级多类型产品性能优化为目标的自有应用性能监测平台，支撑企业全部产品性能监测、分析、优化，将用户体验改善日常化、持续化，并不断完善，并企业内全面推广。

如何搭建性能监测平台

性能监测平台主要分为监测客户端体系和服务端，监测客户端体系在前面的各类监测中有详细介绍，这里主要介绍服务端架构，架构如图4-1所示。服务端架构主要分成监控、产品、运营平台、微服务、任务调度、缓存、安全、服务发现、数据管道、静态平台等部分。存储和计算服务集群归于数据管道和静态平台的部分。数据管道中细分成采集器、持久化缓存、数据集成、数据存储、计算等模块。

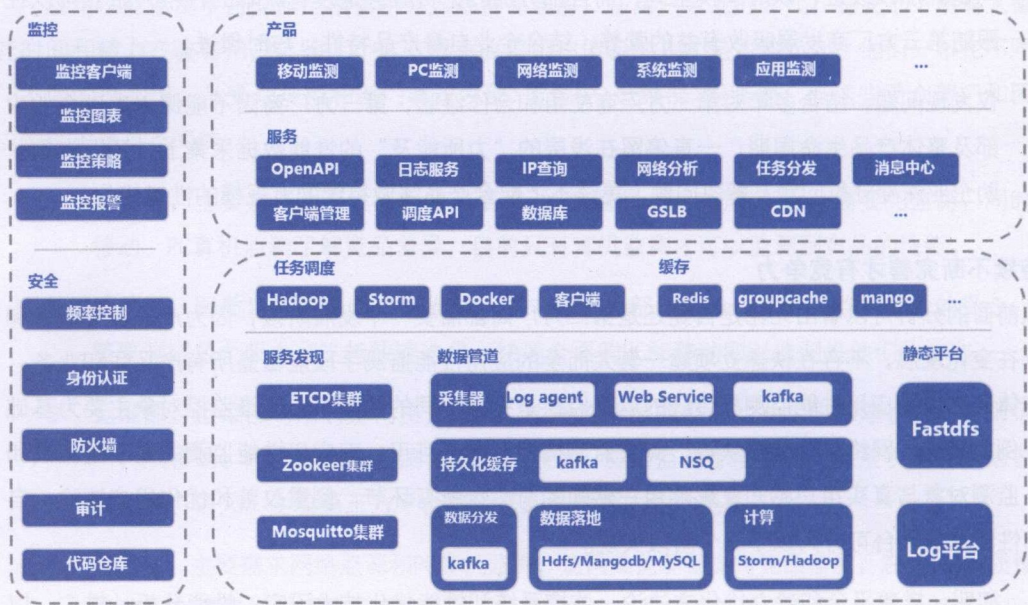


图4-1 监测平台服务端架构图

表4-1 监测平台服务端模块及介绍

模块	作用	实现原理
监控	所有基础平台监控统计	任务调度数据查询统计、认证审计查询统计、监控查询、分布式系统统计查询等

续表

模块	作用	实现原理
安全	对开放API的安全保证、平台的身份认证和审计	API频率控制、平台身份认证、API的身份认证、系统/应用防火墙、代码云仓库保存
服务	所有支撑前端业务的服务端服务，提供对外/对内API	OpenAPI(高级图片处理、CDN操作等)、日志服务、IP地域查询、网络评测服务、计费服务、消息中心服务、数据查询服务、任务调度服务、数据库分表服务等
任务调度	所有类型调度任务的汇集	Hadoop的mapreduce任务的调度管理、storm的流式计算任务的调度管理、所有docker容器的调度管理、所有的pc客户端的调度管理
缓存	所有支撑业务的缓存集群	Redis集群、mango集群、groupcache集群
服务发现	使用分布式配置集群实现服务发现功能	Etcd集群、zookeeper集群、mosquitto集群
数据管道	负责所有数据的采集、持久化、分发、计算和落地功能	<p>主要分成采集器、持久化缓存、分发、落地和计算模块</p> <p>采集器: log agent, http发送客户端、kafka生产者</p> <p>持久化缓存: kafka、nsq</p> <p>分发: 通过kafka消费者进行数据分发消费</p> <p>落地: 消费后的数据落地到Hdfs/Mangodb/MySQL中</p> <p>计算: Storm流式计算, Hadoop离线计算</p>

3.1.1 产品逻辑及用户行为

应用性能与产品设计、开发并序。应用性能当中有数据质量及人为设计缺陷之分。而产品的设计缺陷则基础设计已经相对固定了应用性能的存在。例如网页的功能丰富，交互有多本身就会带来负担，并导致应用性能问题的发生。而互联网的产品相对多发现得越早产品部署上线后，通过心，从而让用户体验产品更流畅。产品优化的过程，也是让用户体验成本更低，并直接提升用户体验。

第3部分 分析、优化篇

产品与用户体验是不矛盾的。产品优化的过程，也是让用户体验成本更低，并直接提升用户体验。

产品逻辑复杂、程序复杂、大量模块、功能复杂、用户产生内容。

产品功能过于复杂、重复、过多的交互方式、交互流程复杂。

产品对外宣传繁多、内容过大、不合理的推广和营销手段，导致用户反感。

从产品角度，需要关注的产品问题有一个原则，一二代产品小的产品问题可以容忍，三代产品大的问题必须解决，四代产品小的问题也必须解决，五代产品大的问题也必须解决。

产品性能分析实践

应用性能优化实践

性能优化平台搭建实践

产品性能分析实践

应用性能优化实践

性能优化平台搭建实践

产品性能分析实践

应用性能优化实践

性能优化平台搭建实践

产品性能分析实践

应用性能优化实践

性能优化平台搭建实践

第5章 应用性能分析实践

5.1 产生性能问题的因素

应用性能是互联网产品的核心竞争力之一，同时也是一个成功产品的必备要素。而产品从研发到运行也如同人的健康一样，在漫长的生命周期中无时无刻地受到内外众多因素的影响。所以改善和保障产品体验已经成为当务之急，同时也是当前互联网从业人员的高阶必修课。尤其当我们身处具有中国特色的互联网世界里，有很多特殊之处，应用性能问题几乎无处不在，所以我们更加迫切地需要了解这些影响应用性能的原因，从而减少应用性能问题的发生，最终达到让应用保持好的性能、让用户享受好的体验之目标。接下来，我们将进一步讨论产生应用性能问题的主要因素：

- 1 产品逻辑及用户行为。应用性能与产品设计、开发共存，随产品“出生”而发生，应用性能当中很多场景是由人为设计出来的。
- 2 中国基础网络是全球最复杂的基础网络。这一基本国情决定了应用性能问题存在的必然性。
- 3 PC端环境。用户入网及PC客户端的“野蛮入侵”桌面，导致很多PC环境下的应用性能被强加。
- 4 移动端环境。移动网络、移动软硬件及厂商之林立让移动应用性能与生俱来。
- 5 代码及应用。不仅前端是应用性能最直接的生产者，后端及各类开发语言也是。
- 6 服务器及云环境、硬件、操作系统是应用性能最终的载体，同时也是应用性能的“温床”。

5.1.1 产品逻辑及用户行为

应用性能与产品设计、开发共存。应用性能当中有很多场景是人为设计的出来的，而产品的各版本功能及UI设计已经相对决定了应用性能的存在。例如网页的功能丰富、交互繁多本身就会形成负担，并导致应用性能问题的发生。而互联网的产品领袖多次倡导将产品做得小白化、傻瓜化，从而让用户体验产品更自然。不管是在腾讯，或是在百度都能经常听到“产品要能让孩子轻易上手”，“产品要让清洁阿姨都会使用”之类的言论，可见太过复杂的产品其本身就是一个“BUG”，不仅会人为拉高用户使用门槛，也会让用户使用成本更高，进而直接影响用户的产品体验。

产品与用户体验是矛盾的

- 1 产品本身的功能逻辑与用户使用时的理解存在天然隔阂，两者之间要达到平衡需要长时间优化和用户使用习惯的培养。
- 2 产品逻辑复杂。秒杀活动、大规模推广、高峰期访问、用户产生内容。
- 3 产品功能规划过于丰富。富媒体、过多的第三方及客户广告等拖累速度。
- 4 产品对外合作繁多、内容过大、不合理的推广加重系统负载，导致速度变慢。
- 5 产品在项目前期的强势和急于出成绩会让开发线忙于功能，疏于架构和程序稳定，这也是速度慢的原因之一。
- 6 产品设计过于绚丽、丰富，元素过多，素材追求高质量等影响速度。

用户自身主观感受

毫无疑问的是，人们对“用户体验”的理解几乎是一致的，即人的直观感受。虽然每个人对事物都有自己的理解和反应，但对互联网熟悉的用户与对互联网陌生的用户在使用产品的思维模式上是完全不同的，耐心不同的用户也是一样。举例来说，耐心好的用户能忍受应用打开慢一点，而耐心不好的用户则会直接关掉。有时甚至太过刺激的颜色也会让用户产生反感。所以说，在产品的研发和开发过程中，始终需要注意用户偏好，尽可能规避让用户产生不悦的场景。

5.1.2 中国基础网络

世界最复杂的基础网络

众所周知的是，中国具有世界上最复杂的基础网络。有人甚至调侃说，世界上最遥远的距离就是电信与联通之间的距离。细想也不无道理。正是由于运营商众多而又各自为政，才形成如今这多

网割据、互通成本高的局面。所以说基本国情决定了最终用户和骨干网络具有其各自特殊属性。只要有做过国际化产品的人应该都知道：国际化产品在各国落地，就如同在中国的某个省份落地一样简单。网络层是介于用户层、应用层与系统层之间的层次。也就是说，如果网络层出现波动，将直接影响用户体验。而严重的网络故障，甚至可以导致产品长时间不可用。接下来，我们将进一步探讨，通常的IDC故障是由以下几点原因引起的。

- 1 历史遗留的瓶颈。由于运营商基础架构更新慢，且带宽交互仅在北京进行，所以导致日常交互带宽利用率均在90%以上，系统几欲不堪重负。而骨干网络电信、网通在繁忙时段平均利用率也超过80%。由于负荷重，所以上层链路拥塞乃家常便饭，而跨运营商问题也是层出不穷。
- 2 突发性、人为性。由于各项业务规模从小到大（数千台服务器），所以不可避免地会出现跨机架、跨交换机、跨机房、跨城域网络等问题。而网络设置升级变更及施工则更加容易造成网络故障。在过去工作中，我就曾遇到过数次由于施工、修路或挖断光缆等原因造成的网络波动和中断。
- 3 政治色彩。在重大事件前，如奥运、亚运、每年的两会等，都会对网络做较大的调整和过滤，造成网络不稳定。

用户分布及入网个性化

基本上，用户的本地入网与服务器所在地、所在的运营商已经决定了用户使用产品的体验好坏。如果电信用户跨网访问到联通IDC中的应用，要比正常访问慢3倍以上。而互联网产品是绝对追求用户至上。为了解决这一问题，我们的投入是国外互联网企业无法想象的。这也可以回答为什么国际云服务商是以国家为单位建立数据中心，而在中国则是以省或城市为单位建立数据中心的原因之一。当然这也是国内CDN行业如此繁荣的原因之一。

云数据中心分布局限

就目前情况来讲，国内外的云厂商（包括国际著名云厂商）都选择在中国建立云数据中心，也依旧以大区为单位。实际上，由于云数据的发展阶段及规模直接决定了数据中心的分布及数量有限，所以即使是多线BGP接入，也依旧会有局限性。

国内国际互访困难

国内骨干网络与国际网络交互历经多年，依旧成本高昂、效率低下。而互联网产品想要国际化，更是要经过亚洲、北美、欧洲等地区的层层中转、突破，并且互访延时是国内的数十倍。

国内外网络攻击频繁

随着我国互联网的不断深入发展，网络安全问题也日渐突出。它已经成为了互联网领域的一个重大课题。而网络攻击的手段更是层出不穷，并且不断更新。现如今，300~500GB规模的网络攻击已经成为当下常态。所以说，网络攻击已经成为影响应用性能的重要原因之一。

5.1.3 PC端环境

接入网络复杂

在中国，由于用户分布本身具有地域性，并且本地接入网络也具有运营商属性这两点不可抗因素。所以用户想要获取好的产品体验，除了自身需要本地网络为主流运营商外，访问产品的服务器分布距离够近也是不可或缺的。事实上，优秀的互联网企业都基本上做到了这一点。像BAT就基本实现了五大区IDC覆盖以及全国300+静态CDN节点覆盖。但可以肯定的是，国内600万+中小型企业网站中绝大部分是没有条件做好的。从客观上看，由于相对应的用户地域分布不同以及接入本地网络运营商差异，这其中得到的体验也是不一样的，特别是针对小运营商而言。

用户端硬件配置

Web 2.0时代的来临，意味着产品形态更丰富，交互更频繁。而用户电脑硬件的新旧及配置的差异，使其在使用同样的浏览器访问相同的应用时，得到的体验也是完全不一样的。所以当我们做性能监测的时候，会将慢速或异常的用户本地操作系统上的硬件资源消耗和硬件配置等信息一同传回到服务端，以便进一步分析出现异常是否与本地硬件以及资源消耗有关联。在大多数情况下，答案是肯定的。部分用户访问异常、慢速与本地资源消耗都有直接关联。

浏览器属性

由于浏览器不同和版本访问内容的区别，以及受自身渲染机制不同的影响，所以最终展现给用户的视觉感受也是完全不同的。例如IE浏览器与Chrome浏览器的访问体验是完全不同的。这是由于访问Chrome浏览器需要更多的资源，而在中国小白用户却占绝大多数，甚至还有相当大比例的IE6用户。而就目前而言，桌面“混战”的当下，各种软件厂商浏览器以各种渠道抢占PC桌面，但也因各家实力参差不齐，直接影响到用户体验产生差异。

系统环境干扰

众所周知，操作系统是管理和控制计算机硬件与软件资源的计算机程序。不同的操作系统及版本来源会直接影响到用户访问产品的体验。特别是在各PC电脑厂商、各破解版、旧版Windows操作系统等混乱局面下，更应该在做性能监测时密切关注这部分慢速或异常用户。

客户端劫持

PC本地客户端劫持问题一直比较棘手。除了运营商劫持，就是发生在PC上的客户端劫持最为普遍。所以运营时，经常会发现用户投诉的内容不是自己产品的内容，而部分异常或慢速的样本内容与监测的实际内容不一样等情况。

恶意竞争

PC客户端商业价值以“暴力”的方式用各种渠道入侵PC用户桌面。想当年腾讯与360“3Q大战”，腾讯曾亮出杀手锏，提示用户“二选一”最为经典。而这之后又出现各种桌面大战，直至最后大多数“小白”用户都在不知不觉中被安装了各种客户端，使得用户在开机启动后，不光加重了电脑的资源消耗，而更进一步影响了用户的访问体验。

5.1.4 移动端环境

终端化是互联网产品将人与信息连接起来的纽带。PC、手机、平板、电视、穿戴设备等都是用户使用产品的载体。由于每种终端入网、显示、操作都不同，所以移动化更依赖于运营商而存在。每当手机用户打开PC端的网页时，WiFi、3G~xG的体验是完全不同的。由于手机加载时间会是PC端加载时间的3~5倍，所以优秀的企业会做智能适配，让用户在不同终端上看到对应终端的产品形式，并通过大数据将不同内容智能推送给终端用户等这些途径来增加用户体验。但真正的挑战其实是移动复杂的生态环境。例如，每当在研发人员控制范围之外出现许多不可控因素时，就会给用户带来非常糟糕的体验。

厂商和机型丰富

在中国，手机、移动设备厂商多如牛毛。大小不一的屏幕、各式各样的配置，各种机型层出不穷，这还不包括山寨厂商。我们可以在想象使用这些各式各样的移动设备时，产品体验也是完全不同的。例如百度移动搜索就要针对移动硬件和入网条件，开发炫酷版、简易版等各种产品形态来应对。尤其是在低端机上应用之时，较高端机而言，更应该采用简易版访问内容才能够保障用户体验。

多移动平台

移动应用程序没有一个统一环境。一般来讲，大多数的移动应用程序都必须要考虑兼容多个厂商和设备。这就意味着要处理各种技术规范、操作系统版本和针对开发版本的问题。众所周知的Android平台以各自为政而出名，每个移动厂商都为其自己的设备定制操作系统。同样，曾经有非常统一标准的iOS平台目前也产生了很多变化。由于iOS应用程序需要支持不同版本，如iPhone

4S、iPhone 5或版本更低的移动设备，以及 iPad 和 iPad mini等。而这些因素却足以让企业在保障移动用户体验面前更加困难。

网络复杂、多变

从第一代到第四代移动通信系统的演进，我们经历了很多名词，例如2G/2.5G（GSM、EDGE、GPRS）/3G（WCDMA、CDMA2000、TD-SCDMA）/4G（TDD-LTE、FDD-LTE）及即将到来的5G。这些网络制式的迭代加上中国移动、联通、电信三大运营商的不同步让移动环境更为复杂，也更加多变。同时，应用性能环境会随着移动信号的改变而变化且更不稳定，所以这从一开始就决定了存在于移动网络中的应用性能是更加具有挑战性的。

5.1.5 代码及应用

架构本质上也是构架。它包含了所有应用及资源构成的集合。也就是说，在一定的设计原则基础上，从不同角度对组成系统的各部分进行搭配和安排，并形成系统的多个功能模块。而这些设置通常直接关乎性能和用户体验。在日常生活中，我们听到最多的是高效、高可用 vs 低效、低可用架构，原因在于这两者提供给用户的体验差别是巨大的。这也就是为什么优秀的互联网企业会致力于孜孜不倦地追求应用性能的原因之一。多少年来，经典的 Web 架构在服务器上用应用程序语言来呈现产品。但随着互联网技术在很多方面正在高速发展中，如更稳定的移动化、更便捷的浏览器、更快速的网络，以及更完善的硬件。而这些变化更带来了开发模式的转变。毋庸置疑的是，全方位的与时俱进可以让应用性能更上一层楼。

在日常工作中，我们常常会遇到的不仅是不处不在的软件BUG、开发语言瓶颈、研发底蕴缺乏、代码质量参差不齐、迭代进程缓慢，甚至是第三方应用性能、团队成熟度和追求等等一系列的因素都会导致应用性能问题的发生。例如：

- 1 架构不严谨。业务发展超越架构支撑能力而导致系统负荷过载，进而导致出现系统崩溃、响应超时等现象。
- 2 不合理、不适用的架构必将影响速度。如单点、无Cache、应用混部署、没有考虑分布式、集群化等。
- 3 研发“功力”和经验不足。开发的APP、Server效率和性能较低、不稳定也是常见的事情。
- 4 没有性能意识。业务上量后系统出现连锁反应，导致性能问题叠加，直接影响用户体验。
- 5 多数的性能问题发生在数据库上。由慢SQL、过多查询等原因将造成的数据库瓶颈，没有按应用读写分离、分库分表。

5.1.6 服务器及云环境

运行应用的硬件和操作系统如同乘坐不一样的交通工具。也就是说，在运行同一个应用时不同硬件和操作系统配合的差别，就如同汽车和火车的差距一样。摩尔定律告诉我们当价格不变时，集成电路上可容纳的晶体管数目，约每隔18个月便会增加一倍，同时性能也将提升一倍。所以说，无论是Google、Facebook还是国内的BAT，全球优秀的互联网企业都在追求前沿硬件带来的性能和成本收益。这一点，光从IDC、机架、服务器、内存、磁盘、网卡等，不同硬件和操作系统上运行应用的性能差距有数十倍就不难看出。

硬件生命周期很短

服务器因老化或其他硬件问题毫无征兆地发生故障是所有人都不愿意遇见的。所以这就是为什么人们常说服务器是消耗品的原因。就国内目前情况来看，BAT三巨头拥有超过100万台服务器，且基本3~4年退役。而退役后的服务器会进入到备用池，其中部分服务器会经过筛选进入内网用做开发、测试机，而另一部分退役的服务器则需要花钱请第三方公司处理。从实际情况来看，腾讯基本是强制性淘汰制。针对旧业务上基本平台或云，或以多台旧服务器换一台新的方式处理。而百度是可选择淘汰制，但仍对旧设备有过渡区间，且基本不超过5年。在百度的运营预算中，每年的支出有很大比例是因替换旧服务器。由此可见，在硬件设备的生命周期中，无论是从性能方面还是从可持续操作上看都是在逐年递减的。

云环境的短板

无论是从技术、安全、稳定、灵活、成本等哪方面考虑来看，云环境都比物理环境更加具有优势。但是，千里之堤溃于蚁穴。一次小小的网络故障，就可以导致云上所有的服务中断。所以说，水能载舟亦能覆舟。网络亦是云的天然短板。尤其在我国的特殊生态环境下，云数据中心所处的地理位置及网络接入的运营商属性也会附加在云环境之上。另外，以数百台服务器的大规模应用集群角度来看，从I/O性能、CPU运算速度、网络吞吐能力等来对比，物理服务器要更胜一筹。

5.2 应用性能分析概述

监测平台主要是针对应用性能数据进行收集和展示，并在不断完善监测数据广度和深度的过程中，逐渐形成的一个系统性评估体系。当然，这也是性能分析的主要目的。从时间坐标来看，性能评估体系最终会反映到监控平台，提升监测平台的质量和完整性。而评估体系主要是建立在用户最终访问的应用路径——从地域<->运营商<->网络环境<->操作系统<->浏览器版本<->以及更多不可控的研发迭代和终端环境之上。也就是说，以上所有环境都可能对用户使用产品造成影

响，而这些影响往往是超出我们想象的。

性能分析基本思路

从Google、Facebook的性能优化经验来看，性能分析将成为我们后续较长时间工作的重点。而我们将由这个阶段开始，对性能分析驱动以及监测平台进行不断的改进。笔者根据个人工作经验，总结性能分析基本思路如下：

- 1 基本原则。以真实用户性能数据为主要切入点，首先关注最主要性能指标，进而往下从终端、应用、网络、系统等各方面作深入分析。
- 2 前期和公司内外团队合作，借鉴他们的性能分析思路和方法。
- 3 调研如何利用数据统计相关知识来增强对性能数据的分析，调研不同产品场景的性能分析评估方法。
- 4 将性能分析的成果集成到监测平台中，并提供自动化分析。

基于足够的样本量

- 1 样本量。完整的性能监测在本书较早的章节中有过详细介绍，故不在此赘述。总而言之，无论是哪一种类型的监测，最终采集回服务端的都是不同时间序列的访问过程样本。事实上，样本的抽样率决定了性能数据代表用户体验的真实度，所以说抽样率越高越好。一般来讲，通常样本量大于10万/天最好。
- 2 不同指标对用户体验的影响权重不同，而不同监测类型的指标也会形成区别。例如网页，白屏时间、首屏时间最高，从网络的角度看延时和丢包最高，从系统的角度可用性权重最大。

5.2.1 从用户及生产环境着手

从用户及生产环境着手是性能分析的主要入口，当然前提条件是这些数据来源于真实用户。通常情况下，我们对用户体验问题的感知是滞后的。最常见的情况是用户投诉或反馈后，技术人员会根据用户反馈的问题进行分析，确认系统是否真的有问题。当然技术人员会最优先确认有没有共性的问题。同时也会从用户环境去确认是否存在用户本地的原因等。而这个过程往往是很被动的。接下来，本人将结合个人工作体会，谈一谈如何主动分析和规避应用性能问题。

从终端维度分析

终端即用户。无论是PC，还是移动端，用户只有通过终端才能使用产品。用户使用产品的始与终都发生在终端环境，从所有原始数据中可以分析出影响应用性能的主要因素，所以监测到终端环

境的所有用户体验过程是最有意义也是最有效果的。

从应用维度分析

应用维度相比终端维度而言要更加靠近用户。这是由于终端维度偏前端视角，而应用维度则主要体现在后端，例如服务器及应用代码层面。假如前端决定了产品的呈现，那么后端基本决定了呈现的逻辑及内容。虽然相比前端来说它的权重重要稍小一些，但这也是应用性能产生的重要环节。

从网络维度分析

网络决定了内容传输效率。如果将同一款产品比做汽车在不同的道路上行驶，那么速度基本上不由汽车本身决定，而主要由路况来决定。网络就是不同的路况，对照国外的网络情形而言，国内的网络局面更为复杂，这也毋庸置疑地成为影响应用性能的重要原因之一。

从系统维度分析

系统是应用运行的载体，也是偏后端视角。系统作为载体也是影响应用性能的原因之一。

5.2.1.1 从终端维度分析

互联网产品的形态是跟随终端演进而逐步产生变化的。也就是说，当用户通过终端使用产品时，当终端设备、操作系统或浏览器发生大的迭代后，产品也需要进行同步兼容，否则将直接导致用户在使用产品时出现异常。所以用户使用的终端及环境决定了产品形态的多样性，而产品形态的多样性也要求我们在监测、分析应用性能时要充分考虑这些个性化特征。

PC端

PC环境相对移动环境而言要简单许多。只需主要考虑在Windows环境下监测即可。而正是由于硬件升级，网络提速，PC环境自身的容错能力要比移动强很多，所以我们通常从以下三个方面进行考量。

- 1 操作系统。如图5-1所示，国内的操作系统99%为Windows，而Mac约占中国电脑设备的0.6%，但每年以15%左右的速度在快速递增。
- 2 浏览器。如图5-2所示，国内PC端浏览器主要以IE内核为主。而随着更多厂商的加入，整体市场的竞争呈现出分散化的趋势。尤其值得关注的一点是，国内厂商做浏览器多数是追求商业回报。这一出发点本身就会影响用户正常浏览，如弹窗广告、网页跳转等。

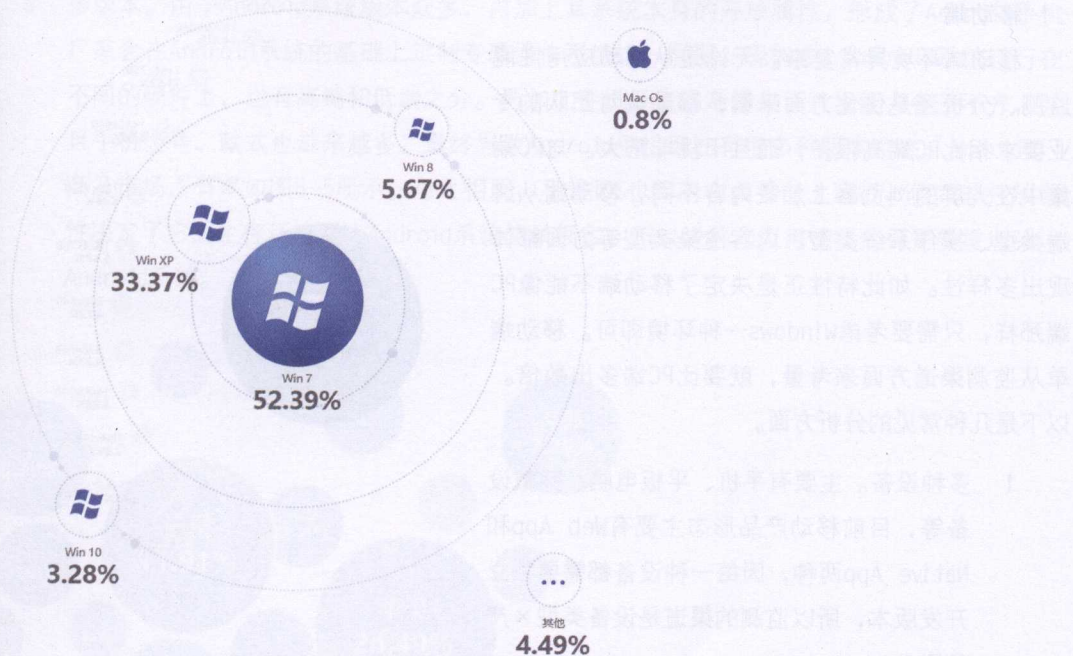


图5-1 操作系统分析

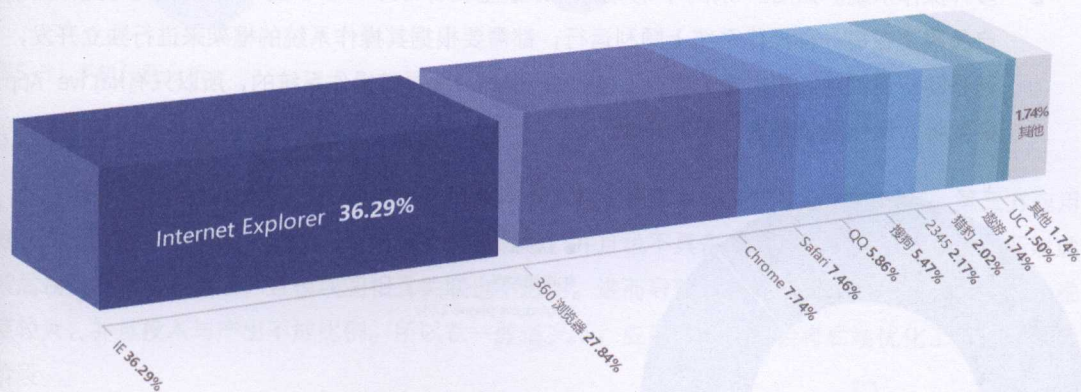


图5-2 浏览器分析

- 3 分辨率。如图5-3所示，分辨率在一定层面上决定了用户的第一视觉感受。由于不同的分辨率决定了浏览器内容填充，进而直接影响前端最核心的性能指标首屏时间。在多数情况下，大分辨率显示器加载的内容多，宽敞，屏数少；而小分辨率显示器加载的内容少，拥挤，并且屏数多。

移动端

移动端环境异常复杂。无论是从移动应用性能监测、分析还是优化方面来看，移动端对团队的专业要求相比PC端高很多，而且干扰非常大。与PC端集中在大屏的浏览器上加载内容不同，移动端从终端类型、操作系统类型、内容渲染类型等方面都体现出多样性。如此特性正是决定了移动端不能像PC端那样，只需要考虑Windows一种环境即可。移动端单从监测渠道方面来考量，就要比PC端多出数倍。以下是几种常见的分析方面。

- 1 多种设备。主要有手机、平板电脑、穿戴设备等，目前移动产品形态主要有Web App和Native App两种，因每一种设备都需要独立开发版本，所以监测的渠道是设备类型×产品形态。

- 2 多种操作系统。如图5-4所示，移动操作系统主要有iOS、Android、WP等。由此可见，如果产品需要在每一种操作系统上顺利运行，都需要根据其操作系统的框架来进行独立开发，而移动应用性能监测也是一样。不过，由于Web App是跨操作系统的，所以只有Native App需要针对操作系统来进行单独开发。

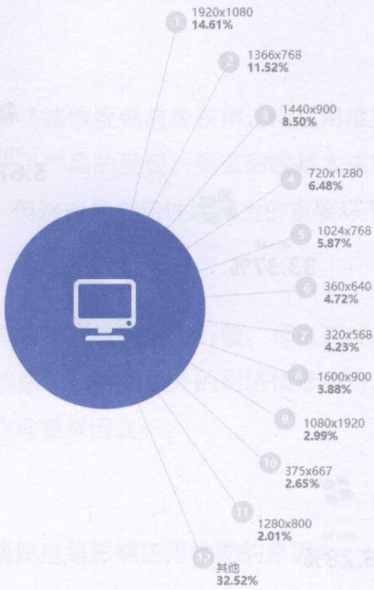


图5-3 分辨率分析

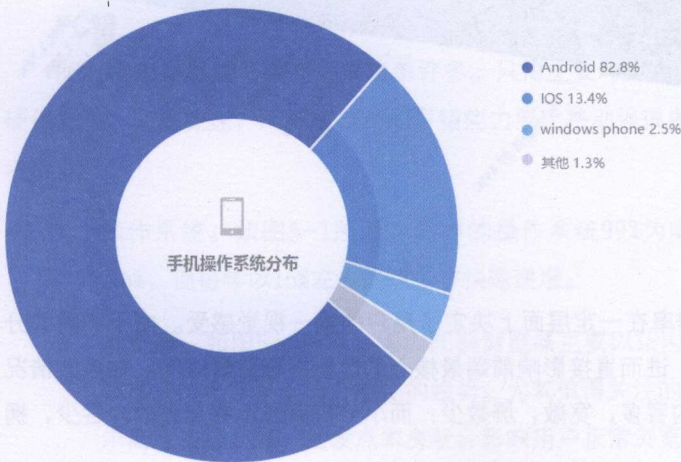


图5-4 手机操作系统分析

- 3 多版本。由于Android系统版本众多，再加上其系统本身的开放属性，形成了Android手机厂家会在Android系统的基础上定制专属操作系统这一局面。现如今，Android系统运行在不同的硬件上，也有高端和低端之分。再加上手机厂商逐渐增加，开发能力参差不齐，并且手机型号、款式也越来越多，最终导致Android手机现在有很多种版本存在，Android厂商及市场占有率如图5-5所示。与之相反，尽管iOS也同时多平台上运行，但其非开放属性决定了它的生存环境要比Android系统简单很多，所以说移动应用性能分析需要重点关注Android系统。

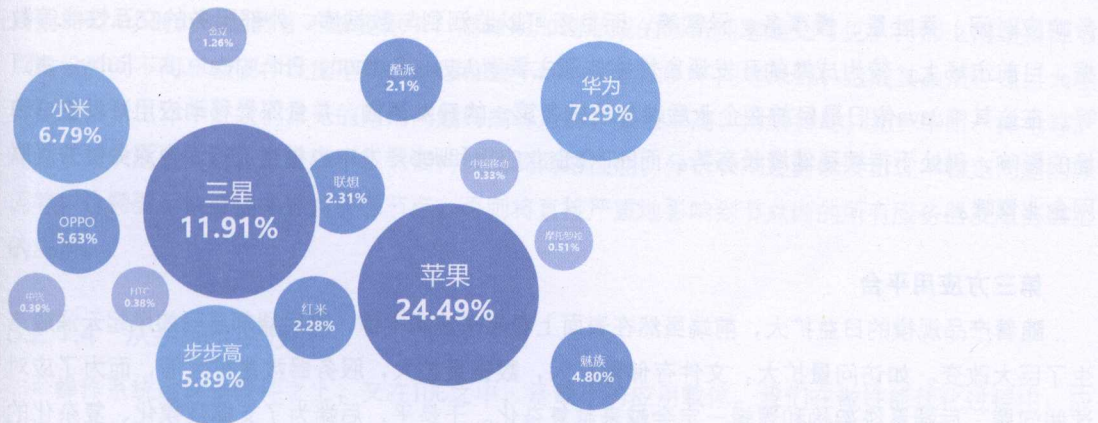


图5-5 手机厂商分析

5.2.1.2 从服务端维度分析

这里的服务端维度主要指后端的所有相关环节，包含系统架构及模块、开发语言、第三方应用等。由于后端不像前端、网络、系统维度那样透明，并且也不具备容易监测和度量的特质，所以通常后端模块繁多又复杂，且模块间相互关联也不透明。进而导致后端性能优化经历的时间较长，难度较大，并且投入与产出不成比例。所以在一般情况下，应用性能优化会将后端优化工作放到最后阶段。

系统架构及模块

由于系统架构及模块本身呈现无规则、多样性特点，促使不同产品的系统架构及模块各不相同。当然，这也形成了系统架构与前端等其他维度非常不一样的特点，那就是缺乏共性。所以要每一个产品后端进行完整监测是不太可能实现的，尤其是针对一些老旧产品。就目前而言，相对清晰的方式就是实行前后端分离。前端使用后端提供的核心模块API与后端进行交互，然后，在用户端可以通过浏览器监测得到核心模块API的响应时间，找出其中响应慢速的模块，同时通过后端模

块程序埋点将整个后端访问流程及响应时间打印到日志中。这种实行两者结合的办法，有助于进一步详细分析，并制定优化决策。

开发语言及逻辑

随着后端开发语言越来越丰富，其形式也越来越多样化，有的适合大型网站，有的适合快速开发，有的适合复杂逻辑，有的适合底层更高性能等。就目前而言，大多数后端都是考虑优势互补，并采取多语言混合模式存在。而从开发语言的角度来看，监测后端性能相对整体架构性能监测而言可行性要高出许多。不光是由于从开发语言的角度可以监测到开发语言自身的性能指标，如Web事务响应时间、吞吐量、慢事务、异常等，而且还可以监测到与数据库、外部服务的交互性能等数据。目前市场上，较为成熟的开发语言性能监测主要有Java、Python、PHP、Node.js、Ruby、.NET等。在这其中Java依旧是目前在企业后端市场排名第一的开发语言，并且深受移动应用开发安卓市场的影响，仍处于持续稳健增长态势。而PHP在企业应用和Web开发中也相当流行，也深受优秀互联网企业青睐。

第三方应用平台

随着产品规模的日益扩大，前端虽然在表面上看起来波澜不惊，可后端却早已如风起云涌般产生了巨大改变。如访问量扩大，文件存储量扩大，数据量扩大，服务器数量扩大等，而为了应对这些问题，后端系统架构和逻辑一定会越来越复杂化。于是乎，后端为了支撑规模化、复杂化的需求势必要进行数据库拆分、反向代理、负载均衡、Memcache、Message Queue、事务处理，CDN、NOSQL、架构升级和Server化等行为，这些举措会令架构逐渐复杂化，并且由于大量第三方应用平台的使用，自然也会带来更多有关性能问题的挑战。就目前来讲，市面上对第三方应用平台监测比较成熟的有MySQL、Apache、Tomcat、Redis、WebLogic、WebSphere、Memcached、Nginx等。

5.2.1.3 从网络维度分析

网络在一定程度上可以理解为是一种资源，它相对于终端与应用维度来说，干扰较少。网络的两端分别是用户终端和应用——用户通过网络访问应用，应用通过网络交付产品价值给用户。从网络分析的角度来讲，也需要将网络本身及两端关联起来。从一定程度上来看，用户端是固定的，那么重点就落在分析应用端所处的网络环境之上。接下来，我们将从以下几个方面进行分析。

从资源类型分析

对于资源型的性能分析，还得从它的主属性出发。网络资源主要有IDC、ISP、CDN、BGP、GSLB等，以上所有类型都能与用户使用产品体验的好坏相关联。每一种网络资源都会经历，从有与无，好与不好，多与少等主要方面进行分析。

地理位置及分布

在国内,网络资源的覆盖效果受到地理位置及分布的影响非常明显,这一客观属性是由基本国情所决定的。也就是说,只有严格按照这个规律来部署网络资源,并合理选择网络资源部署,才能收获性能与成本的双赢。反之亦然。

必须持续分析

网络资源不仅类型多,而且性能不断在变化,再加上近几年骨干网的扩容速度逐渐慢于用户增长需要的速度,所以导致骨干网一直在高负载运行中。同时我们要面临更加严峻的局面是,用户使用的7×24小时网络始终在不断变化中,高峰期加剧形成的网络拥塞随处可见。再加上网络资源节点建设时间不同、软硬件设施不同、地理位置不同、运营团队不同等原因,造成其品质存在巨大差异。在日常工作中,我们常见的网络问题有高峰延时、丢包率高、网络抖动、光纤中断、掉电等。而为了解决这些问题,工程师不仅需要持续监测网络性能,分析长期趋势以及出现不稳定问题的原因等,也需要尽快反馈并更换资源节点,否则将直接严重地影响到节点内的所有服务器及服务器上的应用。

5.2.1.4 从系统维度分析

操作系统运营于硬件之上,又在IDC之中,是最小的应用载体。我们在做性能优化过程中,应当了解和掌握操作系统的当前运行状态。例如系统负载、内存状态、进程状态、CPU负载等信息。因为这些信息是检测和判断系统性能的基础和依据。当然,我们同时也需要掌握系统的硬件信息。例如CPU、内存、IO、网络等状态信息,然后根据这些信息综合评估系统资源的使用情况。值得一提的是,想要掌握应用程序对系统资源的使用情况,需要更深入了解的一点就是应用程序的运行效率。例如是否有程序BUG、内存溢出等问题,并通过对系统资源的监控,就能更快发现应用程序是否存在异常。解决办法是,如果确实是应用程序存在问题,就需要把问题立刻反映给开发人员,继而改进或升级程序。由于系统性能优化本身就是一个复杂且繁琐的过程,所以只有经过充分了解系统硬件信息、网络信息、操作系统配置信息和应用程序信息才能有针对性地展开对服务器本身的性能优化升级。

CPU

CPU是计算机的运算核心和控制核心,是保障操作系统稳定运行的根本,CPU的速度与性能在很大程度上决定了系统整体的性能。

- 1 pidstat。pidstat主要用于监控全部或指定进程占用系统资源的情况。如CPU,内存、设备IO、任务切换、线程等。pidstat首次运行时显示自系统启动开始的各项统计信息,之后运

行pidstat将显示自上次运行该命令以后的统计信息。用户可以通过指定统计的次数和时间来获得所需的统计信息。

- 2 mpstat。mpstat是实时系统监控工具。其报告是CPU的一些统计信息，这些信息存放在/proc/stat文件中。在多CPU系统里，其不但能查看所有CPU的平均状况信息，而且能够查看特定CPU的信息。mpstat最大的特点是：可以查看多核心CPU中每个计算核心的统计数据；而类似工具vmstat只能查看系统整体CPU情况。
- 3 slabtop。slabtop实时的显示内核slab缓存信息，透过/proc/slabinfo（内核的模块在分配资源的候，为了提高效率和资源的利用率，都是通过slab来分配的。通过slab的信息，再配合源码能了解系统的运行情况，比如说什么资源有没有不正常的多，或者什么资源有没有泄漏。Linux系统透过/proc/slabinfo来向用户暴露slab的使用情况。）

内存

内存的大小也是影响Linux性能的一个重要的因素，内存耗尽，系统进程将被阻塞，应用也将变得缓慢，甚至失去响应。Linux系统的内存除了物理内存还包括swap交换分区。当物理内存不足，操作系统会使用磁盘空间虚拟出的内存。

- 1 vmstat。vmstat是Virtual Memory Statistics（虚拟内存统计）的缩写，可对操作系统的虚拟内存、进程、CPU活动进行监控。bi+bo参考值为1000如果超过1000 wa值较大表示系统磁盘I/O有问题，应该排查系统的读写性能。wa参考值超过20%说明I/O等待严重。
- 2 free。free命令可以显示Linux系统中空闲的、已用的物理内存、swap内存及被内核使用的buffer。在Linux系统监控的工具中，free命令是最经常使用的命令之一。

网络

网络带宽也是影响性能的一个重要因素，低速的、不稳定的网络将导致网络应用程序的访问阻塞，而稳定、高速的网络带宽，可以保证应用程序在网络上畅通无阻地运行。

- 1 sar。sar（System Activity Reporter系统活动情况报告）是较全面的系统性能分析工具。可以从多方面对系统的活动进行报告，包括：文件的读写情况、系统调用的使用情况、磁盘I/O、CPU效率、内存使用状况、进程活动及IPC有关的活动等。提供了18种不同的语法选项显示网络信息包括DEV, EDEV, NFS, NFSD, SOCK, IP, EIP, ICMP,EICMP, TCP等。可以实时查看各网络接口的接收传输数据包数量、网络带宽、及丢包等错误信息。
- 2 tcpdump。tcpdump用于截获网络数据包进行包分析的工具。tcpdump可以将网络中传送的数据包头完全截获下来进行分析。它支持针对网络层、协议、主机、网络或端口的过滤，

并提供and、or、not等逻辑语句来去掉无用的信息。例如通过tcpdump，可以清晰地看到发起连接、tcp三次握手、数据传输、关闭连接的全过程。主要用于定位在异常情况下数据中断的原因。

- 3 netstat。netstat命令用于显示与IP、TCP、UDP和ICMP协议相关的统计数据，一般用于检验本机各端口的网络连接情况。netstat是在内核中访问网络及相关信息的程序，它能提供TCP连接，TCP和UDP监听，进程内存管理的相关报告。
- 4 nicstat。nicstat实时监控网卡及网络流量。
- 5 mtr。mtr(My traceroute)是一个把ping和traceroute并入一个程序的网络诊断工具。通过命令可以指定路由由每一跳的发送数据包的个数，并以报告形式显示路由图，比较直观。
- 6 lsof。lsof(list open files)是一个列出系统打开文件的工具。通过命令可以追踪到该网络端口现在的运行情况，包括使用该端口的命令，进程pid用户，句柄数，文件名。

I/O

磁盘的I/O性能直接影响应用程序的性能，在一个有频繁读写的应用中，如果磁盘I/O性能得不到满足，就会导致应用响应缓慢。

- 1 iozone。iozone是一个文件系统的benchmark工具，可以测试不同的操作系统中文件系统的读写性能。可以测试read, write, re-read, re-write, read backwards, read strided, fread, fwrite, random read, pread, mmap, aio_read, aio_write等不同的模式下的硬盘的性能。测试的时候请注意，设置的测试文件的大小一定要大过你的内存（最佳为内存的两倍大小），不然Linux会给你的读写的内容进行缓存，会使数值非常不真实。
- 2 iotop。iotop 是一个用来监视磁盘 I/O 使用状况的 top 类工具。iotop 具有与 top 相似的 UI，其中包括 PID、用户、I/O、进程等相关信息。
- 3 blktrace。blktrace是一个可以显示block的I/O详细信息的工具。查看CPU、网卡、tty设备、磁盘、CD-ROM 等设备的活动情况，负载信息。

综合

- 1 iostat。查看CPU、网卡、tty设备、磁盘、CD-ROM等设备的活动情况，负载信息。iostat中util，即1秒钟里有多少比例的时间IO队列是非空的，对多磁盘、多raid的取所有设备平均值，从整体上反映系统的 IO使用情况。
- 2 strace。strace命令是一个集诊断、调试、统计于一体的工具，我们可以使用strace对应用

的系统调用和信号传递的跟踪结果来对应用进行分析，以达到解决问题或者是了解应用工作过程的目的。

- 3 top。top是一个动态显示过程,即可以通过用户按键来不断刷新当前状态。如果在前台执行该命令,它将独占前台直到用户终止该程序为止。比较准确地说, top命令提供了实时的对系统处理器的状态监视。它将显示系统中CPU最“敏感”的任务列表。该命令可以按CPU使用、内存使用和执行时间对任务进行排序,而且该命令的很多特性都可以通过交互式命令或者在个人定制文件中进行设定。
- 4 gprof。gprof是在运行中收集程序的统计信息。程序的运行方式会严重影响统计的信息结果。
- 5 oprofile。oprofile是Linux平台上一个功能强大的性能分析工具。其支持两种采样(sampling)方式: 基于事件的采样(event based)和基于时间的采样(time based)。
- 6 dstat。dstat会动态显示CPU、disk、net、page、system等负载情况。

5.2.2 常见的分析方法

无论是PC、移动, 还是后端应用性能监测, 每一个真实用户的每一次访问过程都可以理解为一个用户体验样本。这里的样本是指从总体中抽出的一部分个体。而样本中所包含个体数目称为样本容量。所有类型的监测最终都是以抽样的方式采集回用户体验样本, 而应用性能分析及可视化都是基于这些样本之上。通过一系列的统计和分析视图对应用性能进行量化和定位。从数据分析的角度, 为进一步掌握数据分布的特征和规律, 进行更深入的分析。除此之外, 还需要找到反映数据分布特征的各个代表值。两者之间的关联如表5-1所示。

表5-1 常见的分析方法

定序数据	定距数据	定比数据
中位数	平均值	平均值
四分位数	众数	调和平均值
众数	中位数	几何平均数
	四分位数	中位数
		四分位数

5.2.2.1 平均值

平均值也称算术平均数, 是统计学中最基本、最常用的一种指标。而平均值就是把这组数相加, 然后除以这组数的个数, 就一个样本分布的简化描述。尽管平均值是描述应用性能最常用的, 但是它并非最佳方法。主要问题是由于平均值对极端值很敏感, 例如样本分布不集中, 有个别样本

数值过大或过小。应用性能平均值代表一个基本情况，一般看平均值的大小，可以看出用户体验是否糟糕。而目前主要分析视图都是以平均值作为基础，例如趋势平均值，省份平均值，请求平均响应时间分布视图等。

应用性能样本从分布的角度看有很多方式。无论是打散还是排序看样本，最重要的是看样本分布最集中的比例。样本的分布比例通常有以下三种：

- 1 左偏分布。如图5-6所示，这种分布是最理想的，越靠近左边数据值少，快速样本占比越多，但仍然有较多长尾的慢速用户。

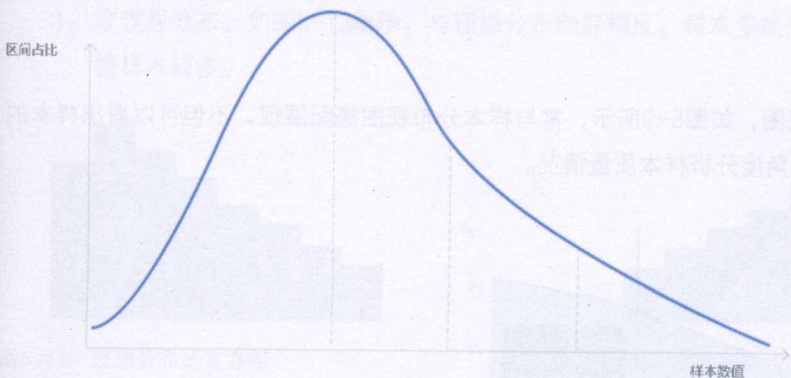


图5-6 左偏分布图

- 2 对称分布。如图5-7所示，这是一个较平均的分布，绝大部分样本都集中在中间，快速比和慢速比都不多，不好也不坏。



图5-7 对称分布图

- 3 右偏分布。如图5-8所示，这是最糟糕的分布，慢速比较多而且集中，快速比较少。

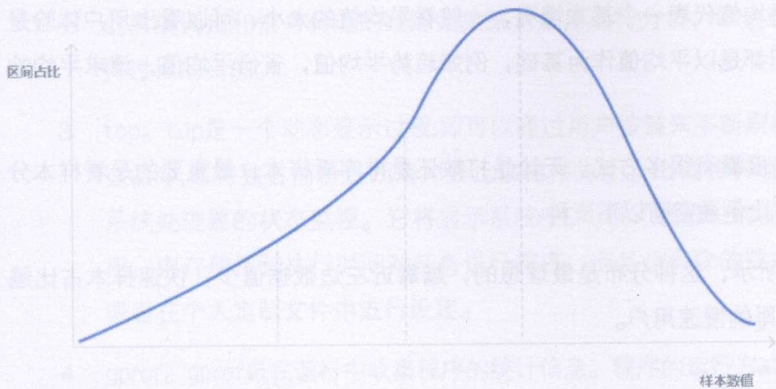


图5-8 右偏分布图

还有一种样本累计面积图，如图5-9所示，常与样本分布视图搭配展现，不但可以看出样本的区间分布，还可以从累计的角度分析样本质量情况。

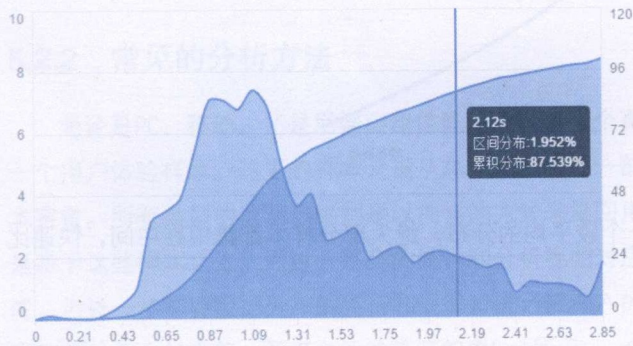


图5-9 累计面积视图

5.2.2.2 几何平均值

几何平均是一种常规的平均方法。是求一组样本数值平均数的方法之一。若有N个数，则这N个数的积开N次方就是N个数的几何平均值。此方法适用于对比率数据的平均，并主要用于计算数据平均增长或变化率，在统计研究中常用以计算平均发展速度。

5.2.2.3 直方图

直方图又称质量分布图，表示性能样本数据分布情况的一种主要视图。用直方图可以直观看出样本数值的规则性，能够比较容易地看出样本性能整体分布状态。

- 1 平均分布。如图5-10所示，平均分布代表适中，多数样本分布在中间，不好也不坏。

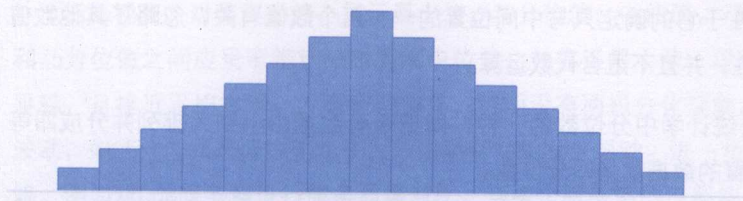


图5-10 平均分布的直方图

- 2 理想分布。如图5-11所示，绝大部分分布在数值小的左边，说明样本数值多数偏小，访问快的样本较多。
- 3 非理想分布。如图5-12所示，与理想分布刚好相反，样本多数分布在数值大的右边，访问慢样本较多。

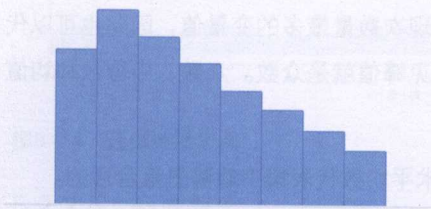


图5-11 理想分布的直方图

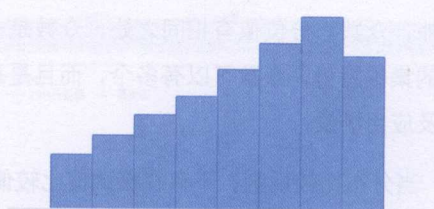


图5-12 非理想分布的直方图

5.2.2.4 分位值

分位值相比平均值表达得更为细致。分位值在统计学中也有很多应用，比如在一般的数据分析当中，我们常用到中位数、四分位数等，需要我们计算25分位（下四分位），50分位（中位），75分位（上四分位）值。25分位代表快速比，75分位代表慢速比。而慢用户比例这部分数据的用户是最需要关注的。分位值主要用来判断样本分布及设定KPI，例如80%的用户首屏时间<1s等。关于分位值视图如图5-13所示，详细介绍如下。

- 1 中位值是一组数据按从小到大排序后，处于中间位置上的变量值。中位数将全部数据等分成两部分，每部分包含 50% 的数据，一部分数据比中位数大，另一部分则比中位数小。由于中位数也是一个位置代表值，其数值的大小不受极大值和极小值的影响，因此中位数具有稳健性和耐抗性的特点。中位数是有序数据值的中间值。它是把数据较高的一半与较低的一半分开的值，中位值和平均值没有必然的关系，中位值是将所给的一组数从小到大或从大到小排列。

- 1) 中位数位于依序排列数值的中间位置，不易受极端值的影响，是较稳健的集中趋势测量指标。

2) 中位数的不足之处在于它的确定只与中间位置的一、两个数值有关, 忽略了其他数值的大小, 缺乏敏感性, 并且不适合代数运算。

2 四分位数。四分位数是统计学中分位数的一种, 即把所有数据由小到大排列并分成四等份, 处于三个分割点位置的数据就是四分位数。

1) 第一四分位数, 又称“下四分位数”, 等于该样本中所有数据由小到大排列后第25%的数据。

2) 第二四分位数, 又称“中位数”, 等于该样本中所有数据由小到大排列后第50%数据。

3) 第三四分位数, 又称“上四分位数”, 等于该样本中所有数据由小到大排列后第75%的数据。

另外, 众数与分位值有相同之处, 众数是一组资料中出现次数最多的变量值, 因此也可以代表数据的集中趋势, 众数可以有多个, 而且是正值, 例如常见峰值就是众数。众数、中位数和均值的特点及应用场景:

1 当分布比较规则, 不存在极端值比较偏离时, 用算术平均数代表集中趋势是最合适的。

2 算术平均数适用于定据或定比变量; 中位数适用于定序变量, 众数适用于定性变量。

3 对分组资料来说, 用算术平均数是非常合适的。

4 算术平均数包含的信息是最多的、最丰富的, 所有观测值与算术平均数差的和等于 1 , 所有观测值与算术平均数的平方和是最小的, 在数学上是很容易计算的。



图5-13 分位值图

指标视图，对整页时间，展示最小值，25分位值，中位值，75分位值和最大值。其中25分位值和75分位值之间应呈窄带状分布围绕中位值，并靠近最小值，说明大部分用户整页时间指标变化平稳，且接近平均水平，没有宽幅震荡，说明没有两极分化现象。监测用户性能是否在大范围内波动，如25分位和75分位差距变大，说明有性能发生抖动。进一步具体定位性能问题地区和运营商等。例如对CDN服务商进行服务质量对比，当两个服务商的性能均值相同时，25和75分位值分化较小的服务商业务更为稳定。



图5-14 分位值趋势图

5.2.2.5 标准差

标准差是一组数据平均值分散程度的一种度量，标准差定义是总体各单位标准值与其平均数离差平方的算术平均数的平方根。反映的是应用性能的离散度。该曲线数值越小，靠近0，表示用户性能越稳定且均匀，趋于均值。反之，数值越大，表面用户性能不稳定或者不均匀，围绕在均值上下波动大。标准差并不反映用户性能是快还是慢，是反映是否稳定。

5.2.2.6 去噪点

去噪点的目的是让数据更少干扰，更准确。通常会将异常的样本及偏离绝大多数样本的最慢、最快的样本根据情况去除。去噪点由于直接决定了最终用户体验的数值大小，所以是至关重要的，并且需要更加谨慎。正确的处理噪点可以让监测数据更健康，如有违之则会适得其反。

5.2.2.7 慢速比

重点分析慢速用户。由于慢速用户在多数情况下极有可能已经发生用户体验问题，所以说一切的性能分析要从慢速用户开始，往往从慢速比中的样本可以分析出最严重的问题。

5.2.2.8 Cache状态

用户浏览器有无cache，得到的监测数据差别巨大。主要有三种状态，有cache、无cache、两

者自然混合状态。正常情况下用户浏览器本地缓存需要第一次访问并且也会过期，同时服务端内容也会随时迭代更新。这些都会导致cache随时处于动态变化中，而且多数情况下是混合状态的。所以，这三种状态的数据都需要监测到并区分对待。

5.2.3 主要分析视图

分析应用性能如同“鉴宝”一样，需要从年代、做工、原料、颜色等各个方面去分析，并将问题列举出来，逐一进行论证和确认。值得关注的是，每当网站或应用发现性能问题后是可以进行快速修复和改进。虽然不同的产品形态，组织和结构差别巨大，但如果单从性能分析的角度来看，各产品的形态是一致的，都可以通过一套科学的分析视图及条件来进行组合分析，通过数十个主流分析视图加上条件组合，可以得到近300个分析视图。

分析视图

无论那一类监测，监测回来的源数据经过处理后，主要通过Web可视化来交互分析，就像操作“Excel”一样。可视化主要按不同视图呈现应用性能数据，目前主流视图如图5-15所示。

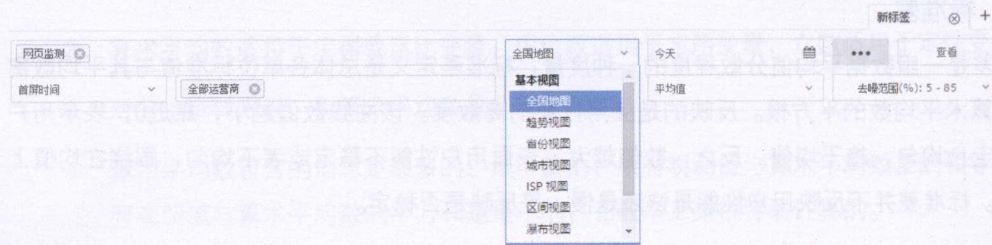


图5-15 视图选项

分析条件

选定分析视图后，需要组合各类分析条件来进行组合深度分析。例如在全国视图中，可以选择不同的性能指标来分析这些性能指标在全国的分布情况，还可以同时选择运营商维度交叉分析，达到分析全国那些省份，那些运营商，那些性能指标最慢等目的。

- 1 性能指标。如图5-16所示，目前主要分为终端及浏览器、网络、服务端、播放器等性能指标，主流性能指标有30个左右。移动和服务端性能指标主要有响应时间、吞吐量等。PC浏览器性能指标主要有白屏、首屏、整页时间等。网络性能指标有首包、DNS、下载速度等。
- 2 运营商。如图5-17所示，相比国外，国内运营商的因素是性能分析必须考虑的，而且不同运营商的性能肯定不同。

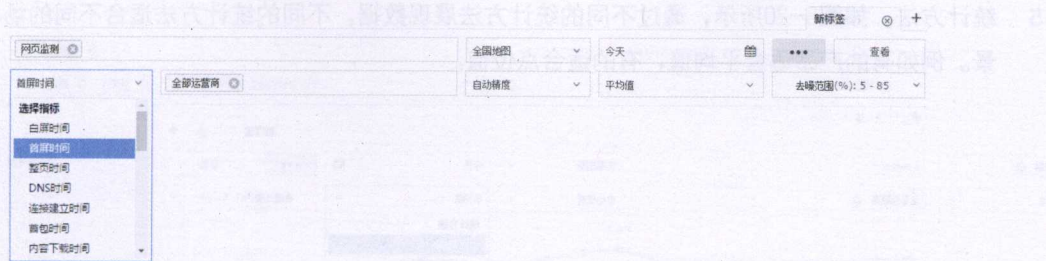


图5-16 性能指标选项

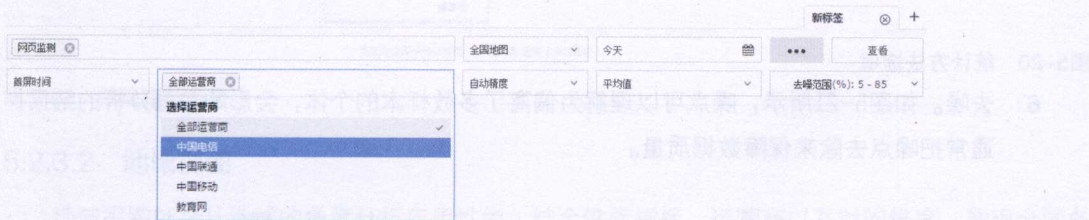


图5-17 运营商选项

- 3 省份。如图5-18所示，省份主要用来分析用户、IDC等地域分布和差异，同时省份与运营商是常用的组合。

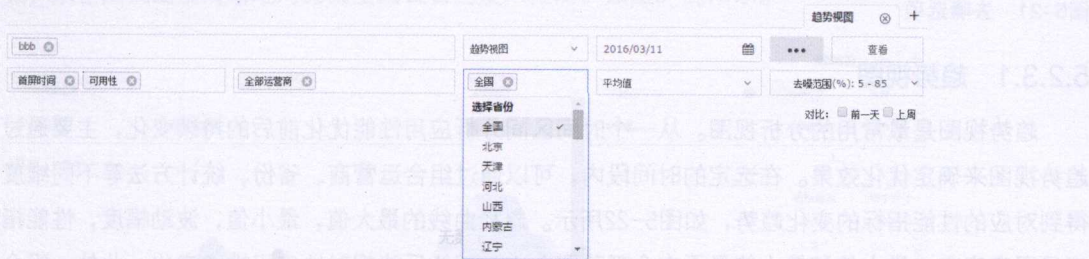


图5-18 省份选项

- 4 精度。如图5-19所示，精度主要用在全国视图，它的作用是将各省的数据差异化，便于从颜色上进行分辨，不具备其他意义。

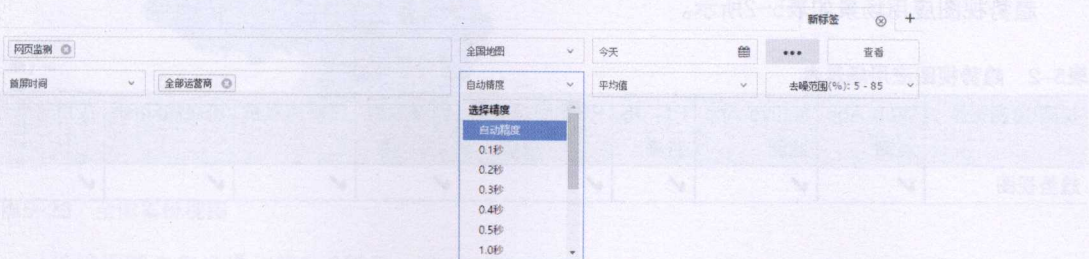


图5-19 精度选项

5 统计方法。如图5-20所示，通过不同的统计方法展现数据，不同的统计方法适合不同的场景。例如有的产品适合平均值，有的适合点位值。

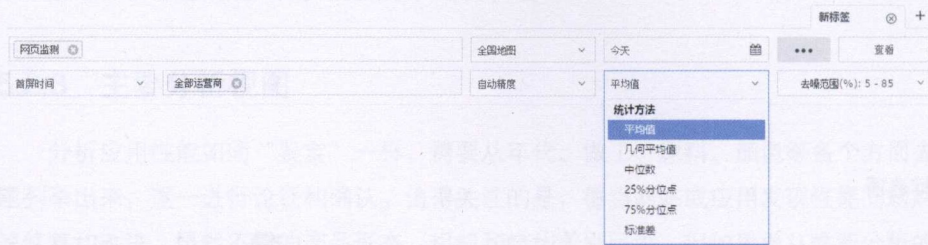


图5-20 统计方法选项

6 去噪。如图5-21所示，噪点可以理解为偏离了多数样本的个体，会影响数据分析的结果，通常把噪点去除来保障数据质量。

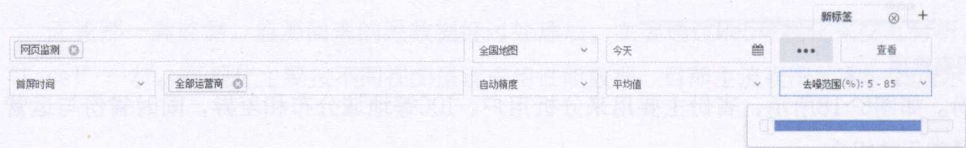


图5-21 去噪选项

5.2.3.1 趋势视图

趋势视图是最常用的分析视图。从一个时间区间分析应用性能优化前后的持续变化，主要通过趋势视图来确定优化效果。在选定的时间段内，可以通过组合运营商、省份、统计方法等不同维度得到对应的性能指标的变化趋势，如图5-22所示。趋势曲线的最大值，最小值，波动幅度，性能指标是否有突变，最大值和最小值是否在合理范围之内，都能反映相对地应用性能变化。此外，组合不同类型的条件有利于从地区，运营商等不同维度，进行省与省之间，省与全国均值之间的对比，以及当天与前一天，当周与前一周的性能对比。

趋势视图应用场景如表5-2所示。

表5-2 趋势视图应用场景表

视图\监测类型	Web App 监测	Native App 监测	PC JS 监测	PC端到端监测	网络监测	流媒体真机	流媒体Flash	应用监测
趋势视图	✓	✓	✓	✓	✓	✓	✓	✓

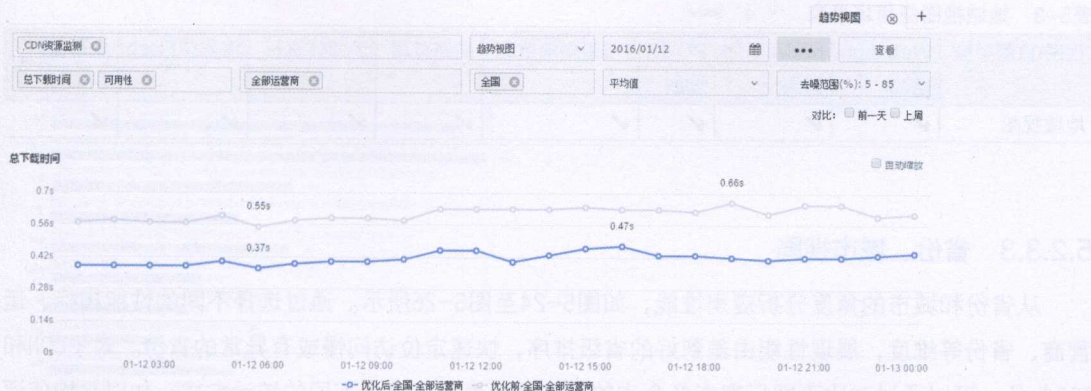


图5-22 性能指标趋势

5.2.3.2 地域视图

地域视图就是从地域的角度分析应用性能，结合性能指标、运营商以及时间维度，展现全国各大区各省应用性能及对比，并快速定位应用性能有问题的地区。随后通过对比各地区的性能数据，衡量全国范围内的服务SLA。另外，通过全国CDN、IDC质量对比，可方便看到被评测CDN或者IDC在全国各地的接入性能。例如进行CDN厂商选择时，可以通过监测多家CDN厂商，通过3~5天的监测数据，从全国视图上清晰地对比出全国各省的接入性能，如图5-23所示。

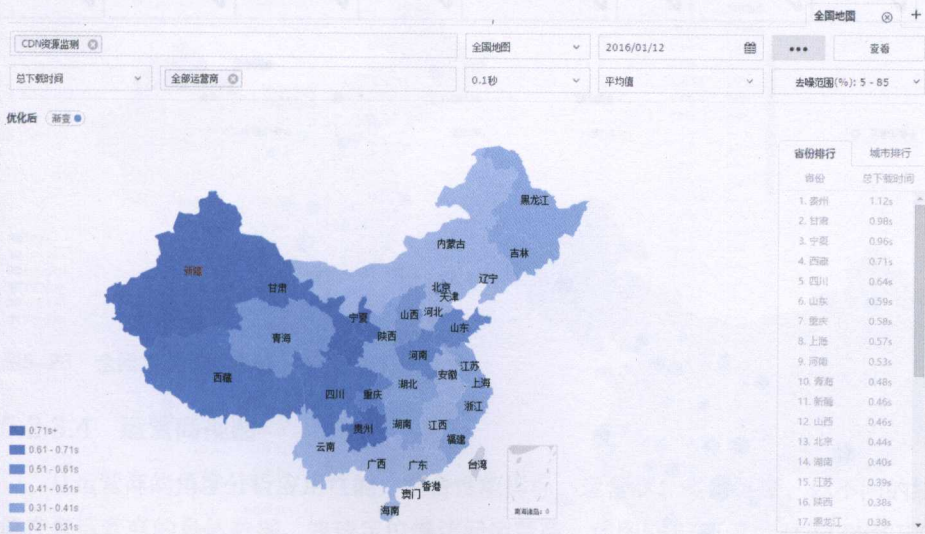


图5-23 全国省份视图

地域视图应用场景如表5-3所示。

表5-3 地域视图应用场景表

视图\监测类型	Web App 监测	Native App 监测	PC JS 监测	PC端到端监测	网络监测	流媒体真机	流媒体Flash	应用监测
地域视图	✓	✓	✓	✓	✓	✓	✓	✓

5.2.3.3 省份、城市视图

从省份和城市的角度分析应用性能，如图5-24至图5-26所示。通过选择不同的性能指标、运营商、省份等维度，展现性能由差到好的省级排序，快速定位访问慢或有异常的省份。对于CDN和IDC业务，可以通过对比不同厂家在各个省的具体性能表现，结合不同的统计方式，如以平均值评估业务速度是快慢，以标准差评估服务是否稳定和均匀。例如对CDN厂家进行服务质量评估，通过省份视图可以对比列出各个省，两个CDN厂家的性能时间差异。同时也可以更进一步对运营商进行筛选，并分别获取各运营商下各个省的性能对比。

省份、城市视图应用场景如表5-4所示。

表5-4 省份、城市视图应用场景表

视图\监测类型	Web App 监测	Native App 监测	PC JS 监测	PC端到端监测	网络监测	流媒体真机	流媒体Flash	应用监测
省份、城市视图	✓	✓	✓	✓	✓	✓	✓	✓

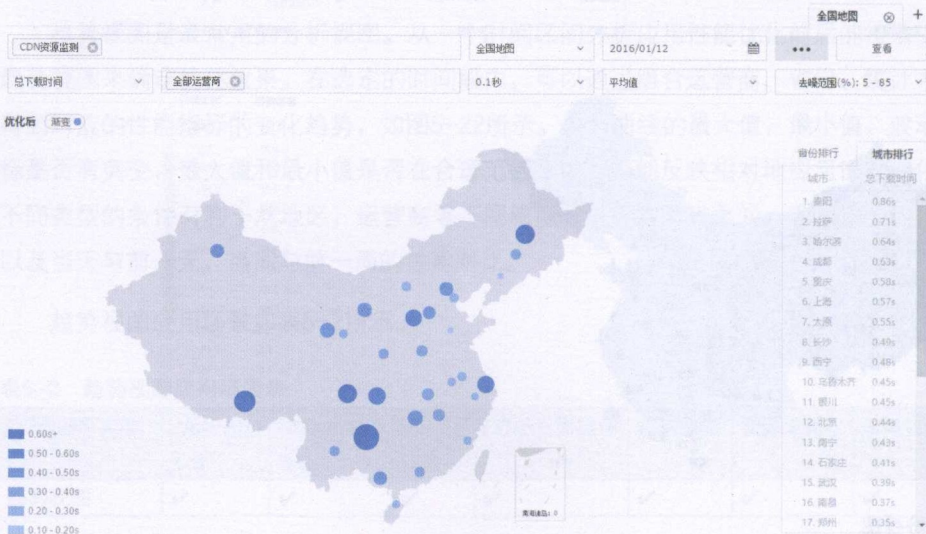


图5-24 省份视图

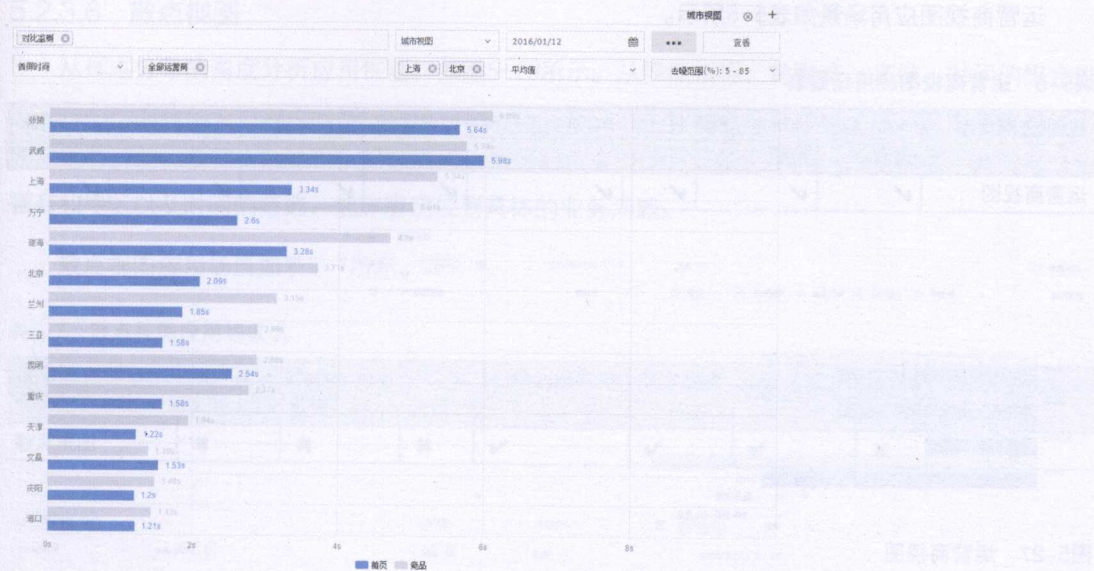


图5-25 城市视图

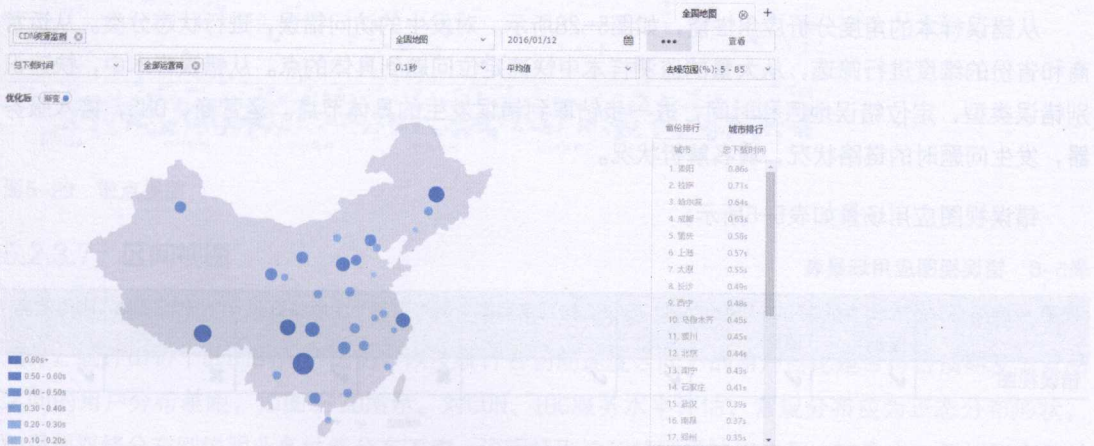


图5-26 全国主要城市视图

5.2.3.4 运营商视图

从运营商的角度分析应用性能，结合性能指标、运营商、省份维度，以不同的统计方式展现性能在各运营商的具体数据，快速定位慢访问运营商，如图5-27所示。对于CDN和IDC业务，可以通过对比不同厂家在各个运营商下的具体性能表现，并结合不同统计方式评估应用性能。例如对CDN厂家在各大运营商下的服务质量评估，可以通过运营商视图，查看全国范围内各大运营商的性能对比。也可以在该视图上进一步对省进行筛选，观测目标地区各运营商下的性能对比。

运营商视图应用场景如表5-5所示。

表5-5 运营商视图应用场景表

视图\监测类型	Web App 监测	Native App 监测	PC JS 监测	PC端到端监测	网络监测	流媒体真机	流媒体Flash	应用监测
运营商视图	✓	✓	✓	✓	✓	✓	✓	✓

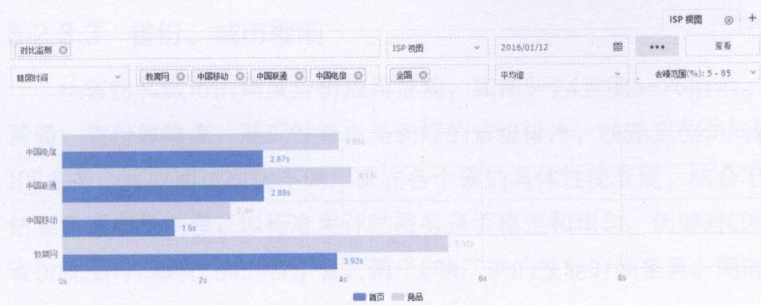


图5-27 运营商视图

5.2.3.5 错误视图

从错误样本的角度分析应用性能，如图5-28所示。对发生的访问错误，进行状态分类。从运营商和省份的维度进行筛选，从大量的监测样本中快速定位问题到具体的点。从错误样本中，快速识别错误类型，定位错误地区和时间，进一步钻取到错误发生的具体节点。运营商，DNS，接入服务器，发生问题时的链路状况，域名解析状况。

错误视图应用场景如表5-6所示。

表5-6 错误视图应用场景表

视图\监测类型	Web App 监测	Native App 监测	PC JS 监测	PC端到端监测	网络监测	流媒体真机	流媒体Flash	应用监测
错误视图	✓	✓	✓	✓	✗	✓	✗	✓

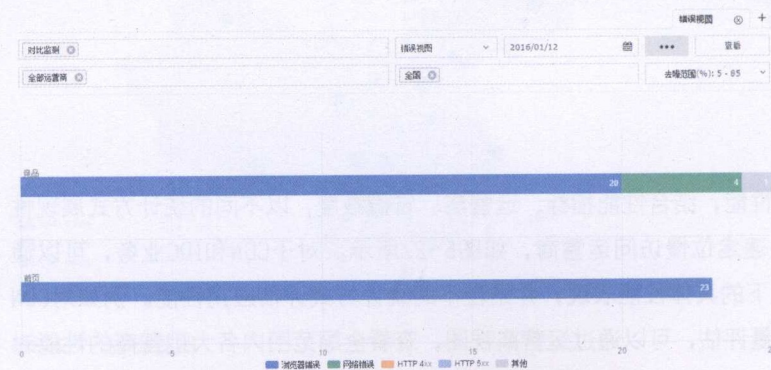


图5-28 错误视图

5.2.3.6 散点视图

从样本分布的角度分析应用性能，如图5-29所示。从性能指标、运营商、省份、时间的组合维度，展示性能指标的监测点分布。通过监测点访问被监测目标的最细节过程信息，对于定位具体问题非常有帮助。从样本的分布，与样本均值曲线的位置对比，并结合不同的监测指标，再通过对运营商和地区以及时间的筛选，能深层定位到具体的业务问题。

散点视图应用场景如表5-7所示。

表5-7 散点视图应用场景表

视图\监测类型	Web App 监测	Native App 监测	PC JS 监测	PC端到端监测	网络监测	流媒体真机	流媒体Flash	应用监测
散点视图	✖	✖	✖	✓	✓	✓	✖	✖

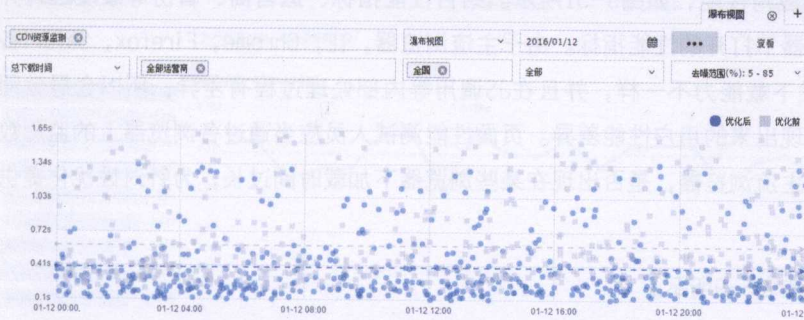


图5-29 散点视图

5.2.3.7 区间视图

从区间的角度分析应用性能，通过性能指标、运营商、省份等维度，将用户访问速度进行区间划分，统计出各个区间的用户比例，以及统计各访问速度区间下的用户占比是否符合预期及与竞品之间的用户分布差距，如图5-30所示。对CDN、IDC服务水平评估，常规分布应为正态分布形状，如呈现双峰分布则说明业务性能分布不均，说明特别快和特别慢的用户都比较集中。例如和竞争对手对比，选取同等大小的文件做对比监测，从区间分布上，可以对比出用户落在快速区间的占比差异，从而体现网站用户体验上和对手之间的差异。

区间视图应用场景如表5-8所示。

表5-8 区间视图应用场景表

视图\监测类型	Web App 监测	Native App 监测	PC JS 监测	PC端到端监测	网络监测	流媒体真机	流媒体Flash	应用监测
区间视图	✓	✓	✓	✓	✓	✖	✓	✖

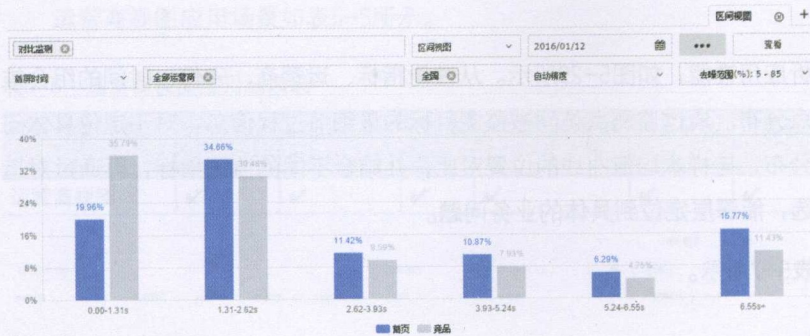


图5-30 区间视图

5.2.3.8 浏览器视图

从浏览器的角度分析应用性能，如图5-31所示。结合性能指标、运营商、省份等维度进行筛选，查看页面在不同浏览器下打开的性能指标。由于主流浏览器，IE，Chrome，Firefox，Safari各自内核不同，HTTP的并发下载能力不一样，并且在JS调用等内部处理过程有差异，所以会导致同一页面在不同浏览器上体现出来的用户性能差异。页面性能测试人员应当通过各浏览器上的监测数据，分析该页面是否适配主流浏览器，是否出现在某些浏览器下加载时间过长，为针对性优化提供数据支撑。

浏览器视图应用场景如表5-9所示。

表5-9 浏览器视图应用场景表

视图\监测类型	Web App 监测	Native App 监测	PC JS 监测	PC端到端监测	网络监测	流媒体真机	流媒体Flash	应用监测
浏览器视图	✓	✓	✓	✗	✗	✗	✓	✗

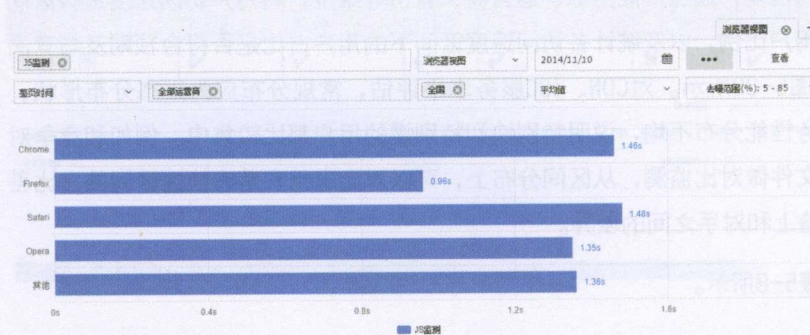


图5-31 浏览器视图

5.2.3.9 操作系统视图

从操作系统的角度分析应用性能，如图5-32所示。从APP安装手机操作系统，展现APP各项性能

指标。统计活跃用户目前使用的操作系统版本Top 5分布；定位性能问题在操作系统版本分布上的共性，为针对App的版本是否要根据操作系统版本进行优化或者APP更新后，对应操作系统版本下性能是否得以优化提供数据支撑。特别针对Android系统，系统版本更新比较快，是否App月活用户都及时进行了系统更新，当大部分月活用户都更新了操作版本后，App自身版本是否能及时更新。

操作系统视图应用场景如表5-10所示。

表5-10 操作系统视图应用场景表

视图/监测类型	Web App 监测	Native App 监测	PC JS 监测	PC端到端监测	网络监测	流媒体真机	流媒体Flash	应用监测
操作系统视图	✓	✓	✓	✗	✗	✗	✓	✗



图5-32 操作系统视图

5.2.3.10 拓扑视图

从拓扑的角度分析应用性能，展示APP或应用调用的各种外部接口的响应时间、吞吐量、错误率，有助于及时掌握调用外部接口的响应时间、调用频率、错误率是否正常如图5-33和图5-34所示。对应的外部服务接口的业务压力是否合理。当对某一接口调用集中，就要关注该接口压力下性能变化。

拓扑视图应用场景如表5-11所示。

表5-11 拓扑视图应用场景表

视图\监测类型	Web App 监测	Native App 监测	PC JS 监测	PC端到端监测	网络监测	流媒体真机	流媒体Flash	应用监测
拓扑视图	✗	✓	✗	✗	✗	✗	✗	✓

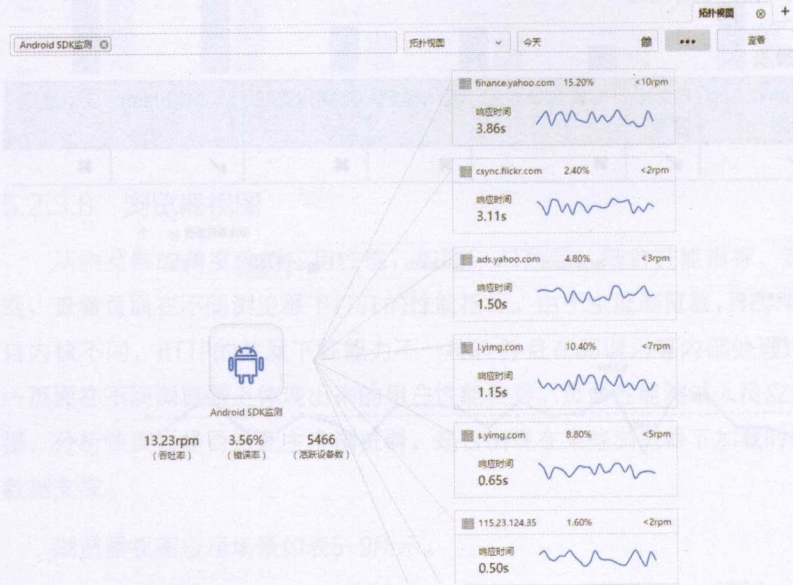


图5-33 移动拓扑视图

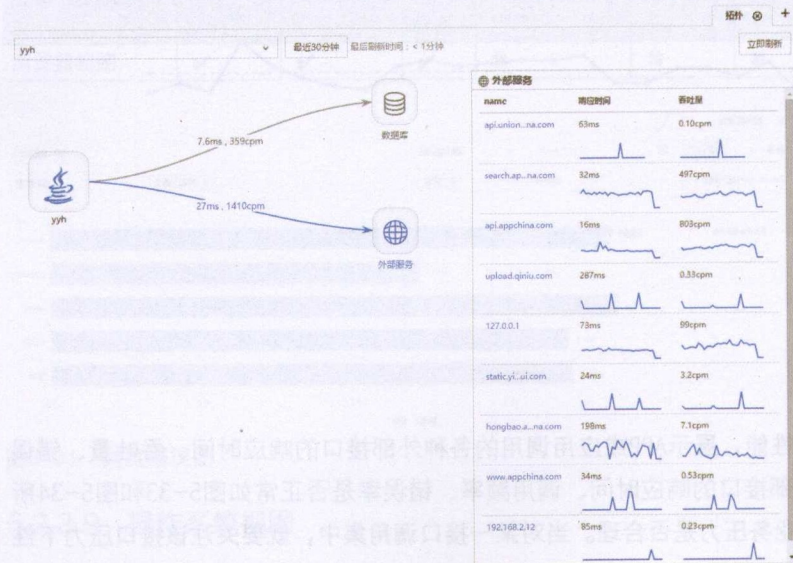


图5-34 应用拓扑视图

5.2.3.11 HTTP视图

从HTTP请求的角度分析应用性能，APP发生的每个HTTP调用的各项性能指标，以时间维度进行展示，如图5-35所示。细化到某个域名下的HTTP请求，特别对关键的HTTP请求的响应时间，吞吐量和错误率是否符合预期。当涉及关键业务的HTTP请求较多，且响应时间下降，则需要定位涉及该HTTP的网络状况，以及服务器的负载能力，是否随着请求增多发生性能劣化，进行相应的压测、优化。

HTTP视图应用场景及视图如表5-12所示。

表5-12 HTTP视图应用场景表

视图/监测类型	Web App 监测	Native App 监测	PC JS 监测	PC端到端监测	网络监测	流媒体真机	流媒体Flash	应用监测
HTTP视图	✖	✔	✖	✖	✖	✖	✖	✖

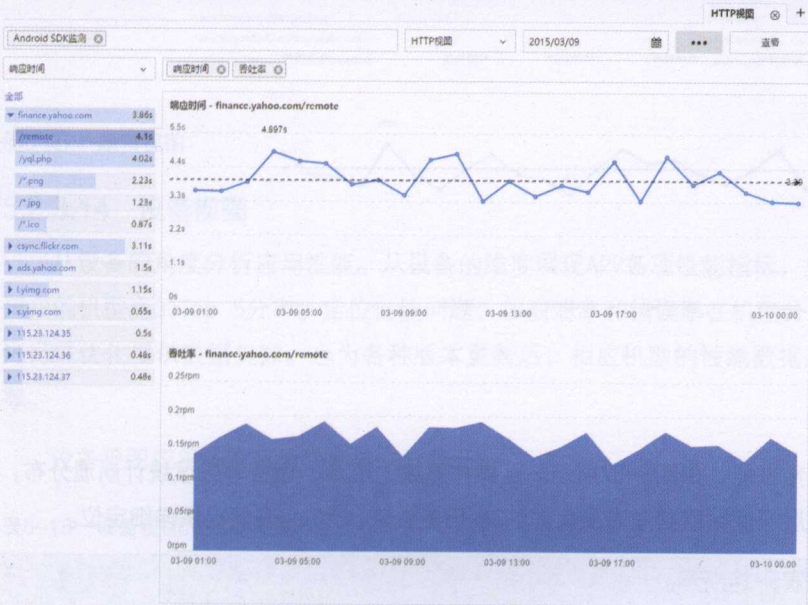


图5-35 HTTP视图

5.2.3.12 交互视图

从应用交互的角度分析应用性能，展现APP内部交互的各项性能指标，如图5-36所示。例如页面的执行时间、交互频率、慢交互列表、前后端线程分析等。定位关键业务的执行时间是否满足预期，统计App的交互热点，结合执行时间，定位是否要进行优化。慢交互列表可以快速定位，慢交互发生的版本，设备，操作系统，当时设备的CPU和内存峰值，以便分析，慢响应是否由APP本身引起，还是由设备当时的性能劣化引起，为开发人员提供慢交互发生时的前端和后端线程的详细信息。

交互视图应用场景如表5-13所示。

表5-13 交互视图应用场景表

视图\监测类型	Web App 监测	Native App 监测	PC JS 监测	PC端到端监测	网络监测	流媒体真机	流媒体Flash	应用监测
交互视图	✖	✓	✖	✖	✖	✖	✖	✖



图5-36 交互视图

5.2.3.13 崩溃视图

从崩溃的角度分析应用性能，如图5-37所示。从操作系统、版本、设备等维度统计崩溃分布，定位崩溃发生的共性，然后深度钻取到崩溃发生前的35s历史轨迹，为App开发提供详细定位。

崩溃视图应用场景如表5-14所示。

表5-14 崩溃视图应用场景表

视图\监测类型	Web App 监测	Native App 监测	PC JS 监测	PC端到端监测	网络监测	流媒体真机	流媒体Flash	应用监测
交互视图	✖	✓	✖	✖	✖	✖	✖	✖

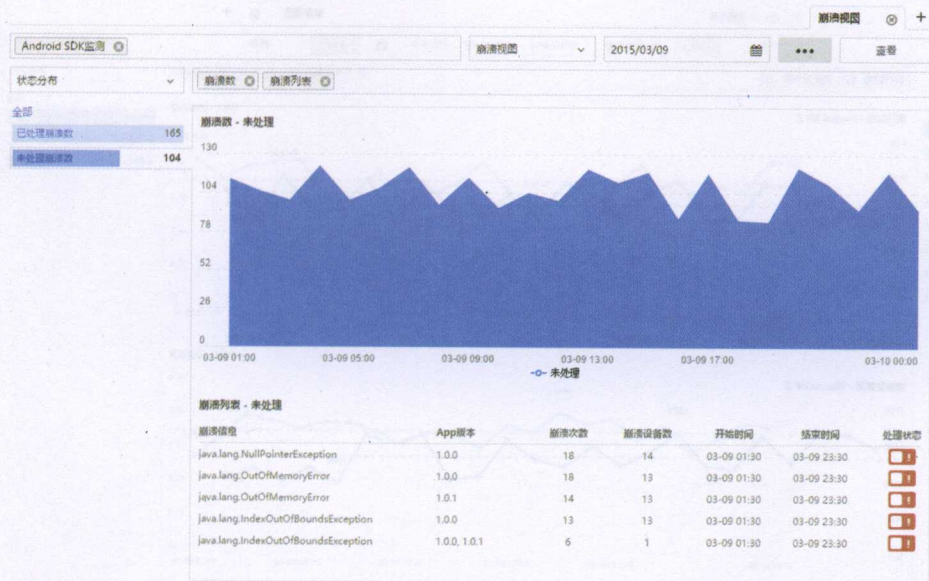


图5-37 崩溃视图

5.2.3.14 设备视图

从设备的角度分析应用性能。从设备的维度展现APP各项性能指标，如图5-38所示，统计活跃用户的机型商的Top 5分布。定位性能问题，如崩溃率和错误率在机型分布上的共性，为针对机型进行性能优化提供数据支撑。也为各种版本更新后，相应机型的性能数据是否得以优化提供数据支撑。

设备视图应用场景及视图如表5-15所示。

表5-15 设备视图应用场景表

视图\监测类型	Web App 监测	Native App 监测	PC JS 监测	PC端到端监测	网络监测	流媒体真机	流媒体Flash	应用监测
设备视图	×	✓	×	×	×	×	×	×



图5-38 设备视图

5.2.3.15 版本视图

从版本的角度分析应用性能，如图5-39所示。从App版本的维度，展现APP各项性能指标。统计活跃用户目前使用版本的Top 5分布，也体现App的更新是否得以顺利发布，以及大部分用户是否及时完成了App升级。定位性能问题，如崩溃率和错误率在App版本分布上的共性，为针对APP的版本升级是否成功以及新版本是否得以优化提供数据支撑。

版本视图应用场景如表5-16所示。

表5-16 版本视图应用场景表

视图\监测类型	Web App 监测	Native App 监测	PC JS 监测	PC端到端监测	网络监测	流媒体真机	流媒体Flash	应用监测
版本视图	✖	✓	✖	✖	✖	✖	✖	✖



图5-39 版本视图

5.2.3.16 Web事务视图

从Web事务的角度分析应用性能，从每个Web事务监测其响应时间均值和吞吐量随时间变化的趋势，并获取慢事务列表，如图5-40所示。统计Web事务吞吐量Top 5，体现出事务热点。对请求最多的Web事务，应评估其响应能力，进一步可以结合服务器监测，当单位时间内发生大量请求时候，硬件资源是否触及服务能力极限，从而进行业务优化或者是硬件资源升级。结合事务响应时间，判断是否有Web事务响应时间持续过长，定位需要优化的Web事务。从慢事务的列表中，直接钻取到响应时间消耗的调用栈列表，定位问题到代码层面。

Web事务视图应用场景如表5-17所示。

表5-17 Web事务视图应用场景表

视图\监测类型	Web App 监测	Native App 监测	PC JS 监测	PC端到端监测	网络监测	流媒体真机	流媒体Flash	应用监测
Web事务视图	✖	✖	✖	✖	✖	✖	✖	✓

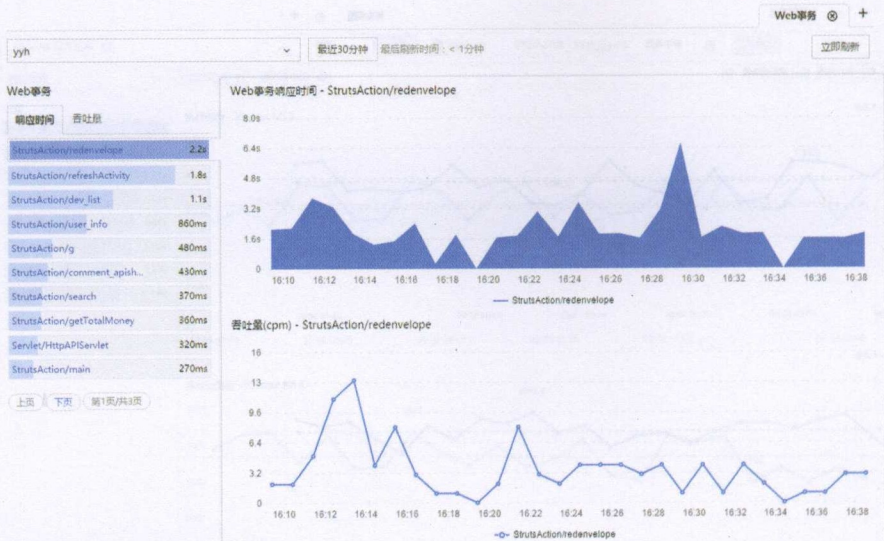


图5-40 Web视图

5.2.3.17 数据库视图

从数据库的角度分析应用性能，从应用服务内部调用数据库语句的角度，监测数据库的响应时间均值和吞吐量，直接列出数据库调用的慢事务，如图5-41所示。统计数据库吞吐量的Top 5，展现数据库对应的表和查询语句的热点。对于吞吐量过高的数据库请求，并评估数据库性能是否能足以支撑业务增长的压力，进一步有针对性地对数据库进行优化。对SQL的慢事务，列出具体调用者，调用次数，耗时时长，调用的具体语句，帮助定位慢查询语句。

数据库视图应用场景及视图如表5-18所示。

表5-18 数据库视图应用场景表

视图\监测类型	Web App 监测	Native App 监测	PC JS 监测	PC端到端监测	网络监测	流媒体真机	流媒体Flash	应用监测
数据库视图	✖	✖	✖	✖	✖	✖	✖	✓

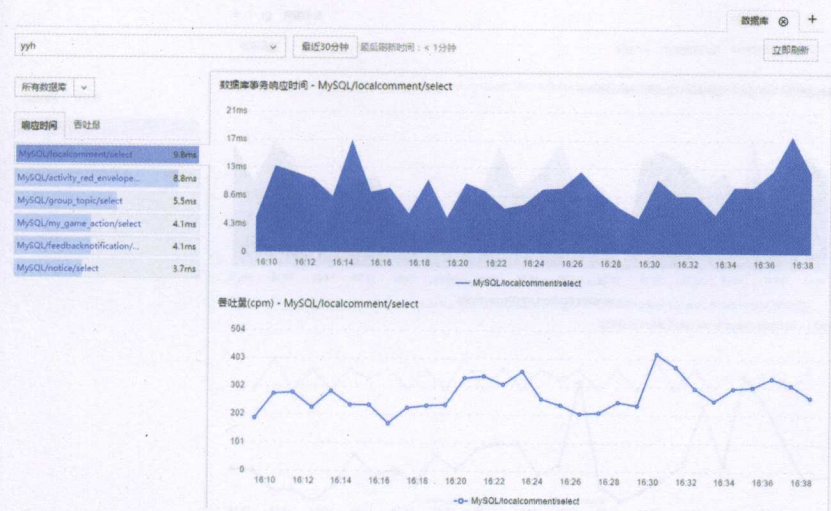


图5-41 数据库视图

5.2.3.18 外部服务视图

从外部服务的角度分析应用性能，应用服务向外部接口调用Web事务的响应时间均值和吞吐量，直接获取慢事务，直观地列出外部服务的接口性能，如图5-42所示。统计吞吐量Top 5并展现外部接口调用热点。对于需要大量调用的外部接口，比如获取图片、在线支付、微信微博转发等，外部接口的性能也会影响本业务的用户体验。对调用量大的外部接口，需要对接口的稳定性，压力承载能力，进行测试评估，响应时间应该保持快速稳定，并且随着业务量的变化，而保持在相对稳定可接受的变化区间之内。如出现较慢的外部接口响应，进一步定位网络层面原因，还是其服务业务逻辑原因，可结合Web监测，建立对外部接口域名，链接或者IP的性能监测，为优化接口提供数据支撑。

外部服务视图应用场景如表5-19所示。

表5-19 外部服务视图应用场景表

视图/监测类型	Web App 监测	Native App 监测	PC JS 监测	PC端到端监测	网络监测	流媒体真机	流媒体Flash	应用监测
外部服务视图	✖	✖	✖	✖	✖	✖	✖	✓

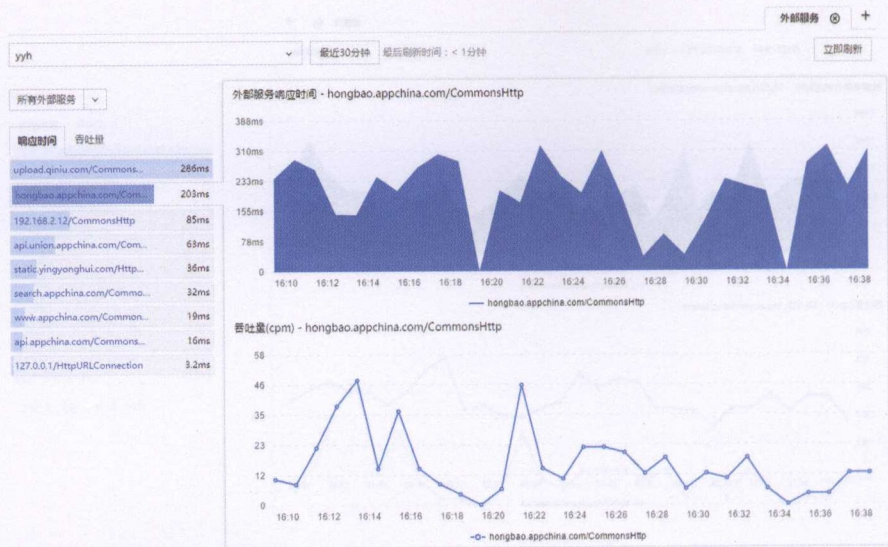


图5-42 外部服务视图

5.2.3.19 后台服务视图

从后台访问的角度分析应用性能，监测应用服务运行在后台的任务列表、响应时间、吞吐量以及系统的CPU和内存的消耗状态，如图5-43所示。在应用服务中，有部分任务是在后台运行。比如定时计划任务，非实时的数据计算和发送，文件清理等。通过后台服务的响应时间，吞吐量，对CPU和内存的消耗，判断在应用服务过程中，后台是否存在异常的任务，会带来资源消耗和服务性能的劣化。

后台服务视图应用场景如表5-20所示。

表5-20 后台服务视图应用场景表

视图\监测类型	Web App 监测	Native App 监测	PC JS 监测	PC端到端监测	网络监测	流媒体真机	流媒体Flash	应用监测
后台服务视图	✖	✖	✖	✖	✖	✖	✖	✓

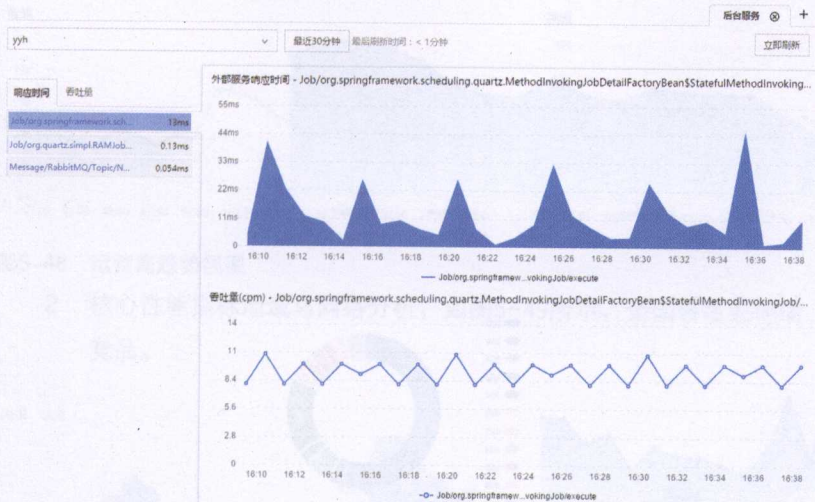


图5-43 后台服务视图

5.2.3.20 其他视图

IDC、CDN的网络流量/带宽、PV/UV也是能直观体现用户体验，需要在服务器部署秒级的系统监测及日志监测Agent，数据传回服务端借助实时的数据分析平台进行数据解析和入库，按产品线及模块、IDC、CDN（第三方CDN服务商，基本能提供日志下载）进行可视化。主要针对最大TOP文件（方便速度和成本优化）、最慢TOP文件（慢速优化）、TOP错误（错误统计直接可以关联到故障）等维度进行统计分析，针对性地进行优化。带宽、流量、用户访问、错误视图如图5-44至图5-47所示。

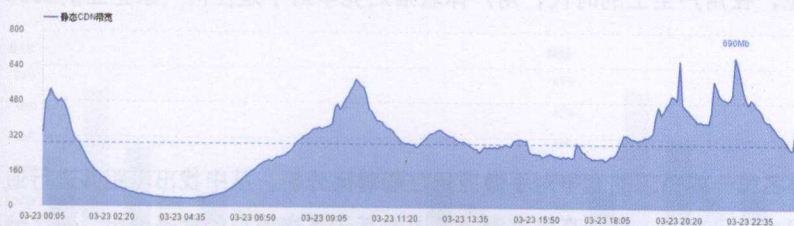


图5-44 带宽视图

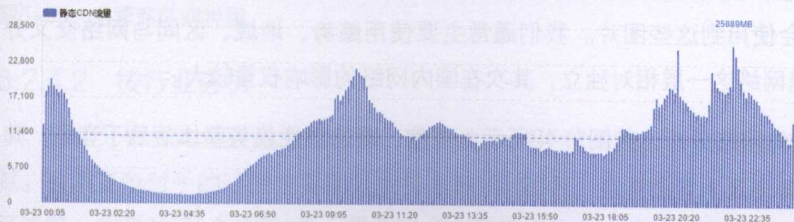


图5-45 流量视图



图5-46 用户访问视图



图5-47 错误视图

5.2.4 横向对比的意义

应用性能对比的意义用一句话直观描述就是“知己知彼，百战不殆”。主要与主要竞争对手及所在行业进行对比。例如通过首页、核心模块、功能页的对比，不但能够清楚了解我们与竞争对手的差距，还可以进一步分析前端、系统、网络、架构等，从中取长补短。并进一步通过不断优化接近并反超竞争对手。以本人在腾讯、百度的经历来看，两位创始人均不吝惜投入时间与精力持续关注性能优化。可见，在用户至上的时代，用户体验落后竞争对手是任何一家企业的BOSS都不能容忍的。

5.2.4.1 按竞品分析

竞品分析主要与行业排名第一或第二的竞争对手做应用性能对比分析，从中找出差距并进行追赶。前面监测章节已经介绍过，只有PC、移动真机监测才可以做基于真实用户的竞品分析，对比分析的内容主要是Web产品形态，例如PC各产品形态及移动Web App等。在前面分析章节已经介绍了众多分析视图，竞品分析也会使用到这些图片。我们通常主要使用趋势、地域、区间与网络交叉分析，突出网络的主要原因是网络这一层相对独立，其次在国内网络的影响权重较大。

- 1 核心性能指标趋势与网络分析，如图5-48所示，电信、联通性能趋势整体落后于竞品，其中联通比电信要快。

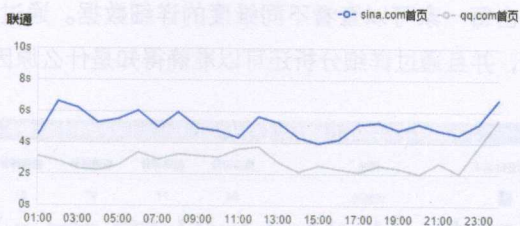
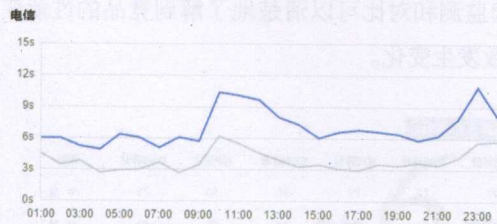


图5-48 运营商趋势视图

- 2 核心性能指标地域与网络分析，如图5-49所示，全国各区域电信、联通整体性能要落后于竞品。

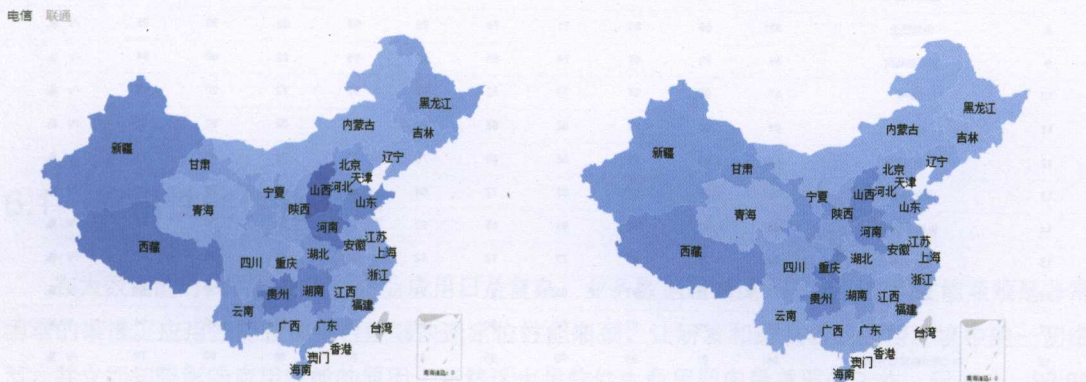


图5-49 运营商地域视图

- 3 核心性能指标区域与网络分析，如图5-50所示，电信、联通快速比、慢速比都落后于竞品。

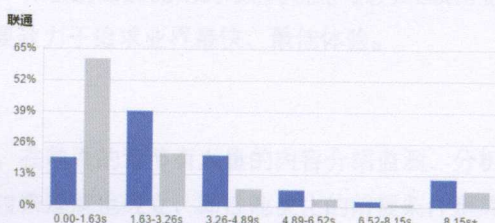
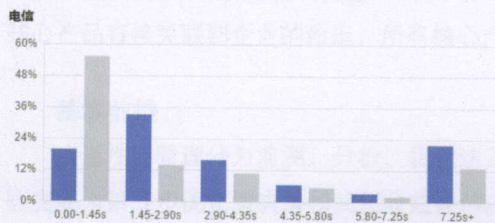


图5-50 运营商区间视图

5.2.4.2 按行业分析

行业分析相比单独的竞品分析，监测竞争对手范围更大，分析维度更多，而且是持续长期跟踪。监测竞争对手的范围主要同行业排名前10~20名，监测维度主要有前端、后端、速度、系统、网络等，监测时间6个月至1年。展现形式主要根据综合应用性能评分排名的方式，如图5-51所示，

点击每一家可以查看不同维度的详细数据。通过持续监测和对比可以清楚地了解到竞品的性能变化,并且通过详细分析还可以准确得知是什么原因导致发生变化。

休闲娱乐	生活服务	综合其他	教育文化	网络科技	行业企业	体育健身	医疗健康	交通旅游	新闻媒体						
行业排名	网站	综合评分	前端评分	后端评分	速度评分	系统评分	网络评分	IDC评分	CDN评分	ISP评分	DNS评分	详细			
1	优酷网	84	77	97	61	91	74	79	70	50	75	▼	点		
2	央视网	89	91	97	83	84	93	95	93	90	94	▼	点		
3	土豆网	89	85	97	76	94	78	80	71	70	76	▼	点		
4	搜狐视频	87	80	97	68	94	85	87	73	90	81	▼	点		
5	爱奇艺	89	92	97	77	92	84	87	67	90	79	▼	点		
6	乐视网	89	92	97	71	91	94	95	94	90	95	▼	点		
7	起点中文网	83	92	94	71	61	93	95	90	90	93	▼	点		
8	电玩巴士	85	89	95	71	76	93	95	96	90	95	▼	点		
9	多玩游戏网	88	88	97	74	86	91	95	92	80	94	▼	点		
10	56网	87	80	97	70	92	88	95	72	90	86	▼	点		
11	酷6网	84	83	97	62	83	90	95	82	90	90	▼	点		
12	17k小说网	85	89	92	68	88	70	63	93	30	75	▼	点		
13	新浪娱乐	83	85	97	66	72	84	86	86	70	86	▼	点		
14	纵横中文网	83	81	96	83	65	92	95	91	90	93	▼	点		
15	腾讯娱乐	84	77	97	73	73	82	86	63	90	77	▼	点		
16	网易娱乐	87	94	97	63	91	94	95	94	90	95	▼	点		
17	4399小游戏	89	90	97	84	80	93	95	92	90	94	▼	点		
18	PPS网络电视官方网站	86	91	93	70	95	81	87	68	90	79	▼	点		
19	PPTV	89	86	97	73	97	92	95	87	90	92	▼	点		
20	Mtime时光网	89	87	94	77	96	67	46	92	40	64	▼	点		

图5-51 行业性能视图

第6章 应用性能优化实践

6.1 应用性能优化概述

在大数据的时代背景之下，企业应用日益复杂、业务数据逐渐庞大，定位应用性能瓶颈是非常困难的事情。应用性能管理平台能够快速定位性能瓶颈，让研发和运维看到应用环境中的一切细节，并立即知晓影响应用性能的原因，当然这也是软件生命周期中最重要的工作。现如今，99.9%或99.99%已经算不上高水准的高可用性了，用户期望的是100%的可用性。为了达到这一点，不仅需要遵循良好的设计模式并保持服务的可扩展性，同时还要保证网络、系统、应用以及硬件等健康运行。最重要的核心是，将所有生产环境中影响用户体验的因素优化到最佳状态。因为互联网企业的核心产品直接关联到企业的命运，所有核心产品要致力于追求业界最快、最佳体验。

基本前提

应用性能管理分为监测、分析、优化这三步，在前面的章节有大量的内容介绍监测、分析，这些主要为第三步优化做决策和优化收益评估。反过来理解也成立，即性能优化的前提条件是具有监测平台及建立在监测平台基础上的数据分析能力，性能优化前后都需要用数据说话。

基本意识

- 1 确保优化方向正确，产品特性和阶段决定优化方向，这是优化的首要条件，优化方向决定了优化的效率和收益。
- 2 确定优化带来的收益，无收益不优化，只有明确收益才启动优化。

- 3 在投入和产出上做取舍，合理的设定优化目标和资源投入，做高性价比的性能优化。
- 4 功能与性能的平衡，产品功能优化是最低成本、高性价比的优化之一，并且立竿见影。
- 5 防止过早和过度优化，过早和过度直接导致人力成本升高以及资源浪费。

整体技术方向、思路

- 1 整理网络、系统、前端、后端，移动等5大方向性能优化最佳实践，针对各个技术细节进行深度调研并在产品线中抓重点问题实践。
- 2 提升工程师在性能优化方面的知识，汇总各个部分的成果，积极组织交流、培训。
- 3 可持续的开发性能优化的工具、库，并集成到基础平台中，提供给产品线使用。
- 4 推广性能优化方法、工具和性能分析指南在产品线中的推广，并指导工程师解决性能问题。

6.1.1 确保优化方向正确

应用性能越来越成为互联网产品运营的重中之重。要提升性能的前提是先进行性能监测及分析，并找出性能的瓶颈，再通过一系列的优化措施进行优化改进。国内外领先的互联网公司，很多年前就开始从各个方面监控和分析性能，并且提出一些切实有效的方案提升了整体性能，我们身边也不缺乏真实的性能优化案例和工具，但最大的问题是如何快速确定优化方向，优化方向通常决定了优化的效率和收益，以便将有限的人力和时间用在“刀刃”上。

首先要达成一致目标

可以很肯定的是，每家企业的每个产品在不同阶段面对应用性能问题的时候，都有不同的理解和出发点。根据我在腾讯、百度经历过较多的情况是产品线各团队忙于实现产品需求，基本无暇顾及性能问题，各团队之间很多情况下也是“各扫门前雪”，更谈不上体系化。所以如果需要彻底地进行性能优化并保持，必须建立在产品、研发、运维、测试等各团队达成一致目标，统一行动的基础上。否则很容易出现前端优化完，收益又被后端的新版本抵消掉，不断的产品迭代会让性能不断出现波动，甚至是恶化。团结能团结的所有力量，共同参与，找主要“矛盾”，不断迭代。另外优化是从做大量日常开始的，最了解产品的人最适合做，要将这部分人尽早吸纳进来。

性能优化≠前端

在很多场合都会听到这样的声音，性能优化主要是前端工程师的事情，但个人认为这个结论更适合早期的互联网时代。在现如今互联网产品多样性、复杂化、全球化趋势下，前端对应用性能的影响已经没有能起到之前的决定性作用了。产品形态、后端、网络、系统、移动、第三方平台等诸

多因素都直接影响着产品的体验，当前的应用性能优化要从多个层次集合形成合力才能达到最佳效果。如果不考虑这些因素并与相关的负责人联动起来，应用性能优化系统肯定是不完整的。

优先解决共性问题

优化是一个长期的持续过程，用最简单的方式达到最好的效果自然最好。但这类的优化很有限，解决同一类问题比解决一个问题更经济、更实惠，优先解决共性的问题可以让所有产品线受益，往往这些共性的问题更基础、更适合早期实现，一劳永逸。而且可以通过复制让更多的产品线享受到收益，常见的具有共性的性能优化方向如下：

- 1 前端，资讯、社区、电商、互动娱乐等产品都是集约化的前端，通过优化统一的模板可以达到优化数量巨大的内容网页。通过统一的前端框架优化，也将惠及所有用户。
- 2 网络，国内运营商及用户入网具有地域性、多运营商等属性，静态、动态、全站、移动、内容型的多元化CDN、BGP加速都可以覆盖所有产品线，并且一上线就将永远受益。
- 3 系统，操作系统是应用运行的载体，也直接影响上面的所有产品，Linux内核、Web Server、应用服务等调优都可以让应用性能上一个台阶，单TCP加速可以让内容传输加快10%以上。
- 4 硬件，硬件之上才有操作系统和应用的存在，BAT服务器生命周期也在3~4年左右，新旧硬件上跑同样的应用，性能是完全不一样的，所以性能优化有一部分工作肯定跟旧硬件打交道。

让所有人感受到价值

每一项优化后，要用数据说话，并肯定参与人的劳动成果，让所有人看到性能优化的收益和价值，让大家保持关注，不断做性能优化项目的迭代。

6.1.2 确定优化带来的收益

用户体验至上的时代，可以肯定应用性能会影响产品访问量、营收，从多个国外咨询机构报告以及多个BAT产品性能优化中也得到了同样的验证。从近几年互联网行业大会也能看到，越来越多的互联网企业开始投入人力、资源在产品的应用性能优化上，部分企业取得了相当大的收获，但同时也走了很多弯路，踩过很多坑，所以在有限的人力、资源和时间的基础上，追求高性价比的性能优化是很重要的。

不见兔子不撒鹰

无收益不优化，只有明确收益才启动优化，而且是优先做收益大的性能优化，确保投入就有回报，

以免浪费时间和资源，这些都可以建立在线下的大量性能测试及有成功经验的其他产品团队之上。

性能优化肯定有副作用

但凡事都有两面性，解决一个性能瓶颈，往往又会出现另外的瓶颈或者其他问题，所以性能优化更加切实的目标是做到在一定范围内使应用的各项性能趋向合理和保持一定的平衡。产品迭代也会打破这种平衡，产品迭代过快，产品复杂度增加，新产品的上线很容易抹杀了前期的优化成果，优化的成果难以得到保持。所以在性能优化一开始就要考虑到各种产生副作用的可能性，一开始就需要规避。

一些验证过的优化经验

应用性能好比我们的健康需要定期进行体检，诊断内科心、肝、脾、肺、肾和外科皮肤、颈、胸、腹、四肢等状态，从而综合得到身体的健康状态，对症下药，采取有效的措施改进。前面章节有介绍速度的监测、分析就是对应用的体检，体检之后需要分类对症优化，解决网络、系统、前端、应用、移动等不同维度的问题，从而达到理想状态。结合在腾讯、百度的优化经验，应用优化是一场持久战，大的产品线需要持续1~2年才可以彻底见效，需要分短期和长期目标来进行，之前的一些相关经验汇总如表6-1所示。

表6-1 验证过的优化经验

最少请求数	最快请求速度	最快可见可用
TAB页异步加载或延迟加载 合并JS/CSS文件 Sprites CSS图像地图 统一公用JS/CSS文件 合并ajax请求 避免重定向 减少iframe请求 头像、图片滚屏延迟加载 不重要的页面模块异步加载 多级Cache	设置长时间缓存 尽量使请求可缓存 Cookie隔离 动、静态应用分离 按类型选择合适WEB服务器 适当使用多域名增加并行下载 设置GZIP压缩 多IDC部署、动态内部代理 使用CDN网络 IP库定位能力提高 图片质量压缩，使用png8 图片预加载 JS/CSS混淆 页面代码压缩 统一公共JS库代码和应用代码 减少Cookie大小 逻辑层协议合并、并行处理 数据、索引内存、SSD并行写	首屏优化原则 减少元素数量、size Html、CSS、JS代码减肥 JS性能优化 Div+CSS布局 优化 避免CSS表达式 Sprites分屏合并CSS、JS 位于页面底部JS并行下载 避免复杂JS循环和计算 统计JS放在页面最尾处 按需加载、异步加载、延迟加载、预加载 后台尽快输出html代码

6.1.3 功能与性能的平衡

在性能优化过程中，大家肯定会遇到这样一个非常棘手的问题，新版本、新功能和新架构调整会将之前努力优化的成果付之东流，也很容易理解，新的功能和改动会打破之前的平衡，结果必然让性能数据产品抖动，也就是让性能更差。做过性能优化的人应该很清楚，如果性能趋势不明显，一会上一会下，其实很容易让优化团队产生迷茫，所以功能与性能的平衡是每个产品团队需要当作重要的事情来考虑的。

功能设计和上线都需做性能评估

各产品线参与性能优化的成员需要把握版本迭代和新功能对性能的影响，特别是在新需求提出前要评估，在需求实现时要避免发生性能问题，在需求实现后上线前需要做上线前的评估。从与历史版本的对比中，如果出现较大的性能退化，需要及时与产品团队沟通，需要整个团队对性能和用户体验有较高的意识。在百度有一种速度准入机制，即发布的页面和移动Web App速度不达标，是不能发布出去的，这种方式适合多人且频繁发布的团队。

产品形态优化比技术优化收益更大

功能都来自于产品经理，也是为了满足用户更多诉求，相同的需求可以使用不同的形式实现，例如资讯类的网站信息和分页很多，如果按产品经理的需求，打开主页或主要频道，是需要加载所有内容的，包括分页内容，其实用户不一定会点击所有分页，所以哪怕延时加载分页的内容，也会让主页加载负担减少很多。这里是需要影响产品经理改变产品形态来实现更好的用户使用体验。根据我在腾讯、百度的体会，很多互动社区、游戏的产品形态也可以从更好的用户体验上去影响产品经理的决策，例如将热点事件的内容事先通过客户端推送到用户本地，用户点击时从本地读取，就不会对服务端产生巨大的冲击。这样通过产品形态的改变达到更好的用户体验往往比技术优化收益更大，代价更小。

6.1.4 防止过早和过渡优化

过早优化会带来更大投入

性能优化是用户体验优化的一种途径，根据以往的经验，稳定型产品最适合做全面整体的性能优化，干扰少，也容易评估收益。初期或快速成长型产品因其快速迭代、与生俱来的不稳定等特性，会造成大量干扰，新版本、新特性甚至会直接淹没掉早期的优化成果，性能抖动较大是不容易形成趋势和可持续优化的。

从开始就需要防止过度优化

应用运行良好的时候恰恰也是各项资源达到了一个平衡体，任何一项过度优化都会造成平衡体系破坏，从而造成人力投入大、投入时间过长或预算投入高等。例如要让一款产品80%的用户在2s能加载完，只要认真去做优化，投入人力、时间和资源是容易做到的，但是要让另外20%的用户在2s加载完，付出的代价会是之前的3~5倍。例如适度使用CPU，如果过度使用会造成大量进程等待CPU资源，系统响应变慢，等待会造成进程数增加，进程增加又会造成内存使用增加等。

所以性能优化项目的深度与适度需要项目负责人来把握，各方向的负责人也需要有过度优化意识，通常我们会将影响性能的所有可能全都列出来，然后分类，分优先级，将容易做、消耗时间少的优化项排在一期先做，然后依次往下排，过程中会平衡，随着优化进展，离我们的目标越来越近，提前协调相关人的时间和必要的资源（例如网络BGP等资源是需要提前做预算，再发起采购），最需要时间和预算的会排在二期，甚至最后，通过这样的方式基本可以平稳。但BAT会有自己的追求，例如在百度，为了加快200ms，投入了3000台服务器（平均2万元RMB一台，共6000万元左右），通过这一方面可以看出，百度在用户体验上是追求极致的。

6.2 网络优化

有一个关于网络性能的小故事，人们铺设一条横跨大西洋连接伦敦和纽约的近5000km的海底光缆。铺设这条光缆的唯一目的是减少城市间的路由，为金融市场的交易商减少5ms的延迟，耗资大约4亿美元。 $1\text{ms} = 8000\text{万美元} = 5\text{亿人民币}$ 。在中国这样复杂的运营商环境、移动互联网快速发展的趋势下，基础网络永远是产品体验的绊脚石之一，特别是在移动互联网时代更加明显，能看到的所有与基础网络相关的名词都可以制约网站速度，在规划篇也花了很大篇幅去介绍网络属性和策略，其中IDC、用户具有区域属性和运营商属性是主要矛盾，正因为有这样的国情才放大了对IDC、CDN就近接入、BGP多运营商接入、GSLB智能调度的依赖，结合个人在腾讯、百度优化的体会，汇总网络优化实践如表6-2所示。

表6-2 网络优化建议列表

网络优化项	优化说明	备注
IDC优化	用户分布及IDC覆盖能力具有区域性，如果简单理解，北方用户和IDC多为联通，南方则为电信，各小运营商分布不均，简单IDC覆盖所在的地理位置周边具有明显优势，按地理位置分单IDC，多IDC，区域IDC，全国IDC，按城市可分为一线、二线IDC。全国分区域的一线城市的多IDC分布具有最大收益	一线北京、上海、广州IDC网络覆盖能力最好，价格最贵，稳定性最高

续表

网络优化项	优化说明	备注
ISP优化	IDC覆盖能力具有运营商属性，同运营商互联互通无问题，跨运营商访问速度要慢3~5倍，同时运营商也具备区域属性，所以针对主流运营商电信、联通及小运营商要进行细分，才可以将网络速度做到极致。个人的体会，目前腾讯网络分布是做得最优势的，全国五大区域14个大规模核心IDC互联互通，58个CDN节点	除主流运营商外，移动、教育网等小运营商用户体验越加重要，特别是移动
CDN优化	中国拥有世界上最复杂的基础网络，也使得CDN成为中国互联网一大特色，CDN既解决用户到IDC最短距离，又减带宽成本、减机架成本、减电成本、减少IDC建设成本等，可以说CDN是速度优化不可缺少的一个环节，速度优化必使用CDN，正确的使用CDN，可以将网络接入速度做到极致	CDN是速度和成本的双刃剑，互联网大公司都选择自建，将这两个价值最大化，可想其重要性
BGP优化	应用区分运营商限制，必须分开IDC部署，主流运营商必须发展区域运营商，但小运营商众多且用户占比小，单独部署不现实，最好的平衡是使用多运营商单IP接入，只要部署一套即可。在当下中国PC网络运营商增加、移动网络风起云涌的时代，BGP的价值将会放大，腾讯、百度都有自建BGP机房	BGP有助于提升移动互联网产品的体验，前提是有专线或内网的保障通信
TDO优化	TDO (Traffic Distribution Optimizer)，可以理解是一种用户定向策略，DNS以TDO为依据，将用户解析到对应的运营商及服务器，保证使用最佳的服务器服务最近的用户，从而确保访问速度，正确的定向可以提升用户访问速度，反之将适得其反。TDO因历史遗留及新入网用户等原因，会存在不能识别情况，要尽最大可能进行识别，准确率要保持>99%，不能识别的需要通过解析策略进行规避	TDO的更新和完善是一个持久的工作，需要自动化和持续投入人力
GSLB优化	GSLB (Global Server Load Balance)，全局负载均衡，包含DNS解析，同时又具备健康检测、测速并智能调度，自动容灾等能力，将用户解析到最快的IDC和服务器，将问题服务器和集群屏蔽和故障自动处理等	GSLB做到最后，对于互联网产品就像人的大脑一样重要

6.2.1 IDC优化

IDC即互联网络数据中心，Google对数据中心的定义是“多功能的建筑物，能容纳多个服务器以及通信设备。这些设备被放置在一起是因为它们具有相同的对环境的要求以及物理安全上的需求，并且这样放置便于维护”，这是较确切的理解。我们日常使用的所有PC、移动产品都依赖IDC并在其中“生根发芽，开花结果”，所以IDC也是决定产品“成长”和用户体验的重要因素。IDC也可以理解为是一种机房资源，向用户提供标准化的数据存放业务以及相关服务，用户可通过数据中心的主机托管、主机租赁、虚拟主机等服务，利用数据中心的力量搭建自己的互联网技术平台。

全国核心节点的特点

全国核心节点之间为不完全网状结构，以北京、上海、广州为中心的三中心结构，其他核心节点分别以至少两条以上链路与此三个中心相连，通过高速数据专线实现国内各节点互联。现在已建成以北京、上海、广州为国际节点和以北京、上海、广州、沈阳、南京、武汉、成都、西安为八大区中心节点在内的，覆盖全国200多个城市不完全联接的多层网络结构，从拓扑结构逻辑上分为两层，即核心层和大区层。

- 1 核心层由北京、上海、广州、沈阳、南京、武汉、成都、西安等8个城市的核心节点组成，如图6-1所示。核心层的功能主要是提供与国际Internet的互联，以及提供大区之间信息交换的通路。其中北京、上海、广州核心层节点各设有两个国际出口，负责与国际Internet互联，以及多台核心路由器与其他核心节点互联，其他核心节点各设多台核心路由器。

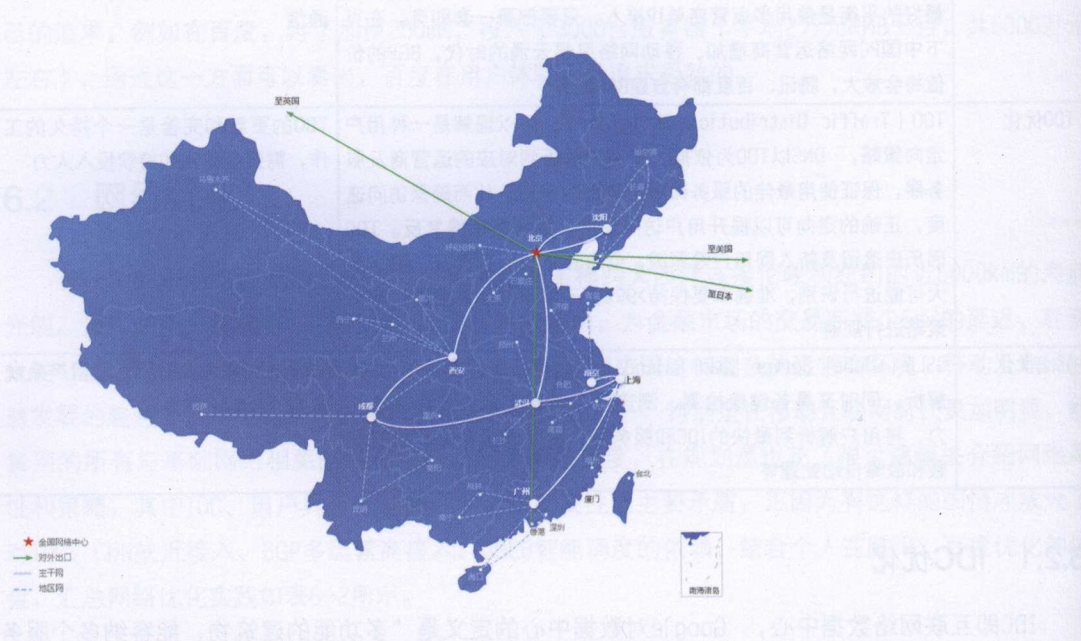


图6-1 核心层视图

- 2 大区层由全国31个省会城市按照行政区划，以上述8个核心节点为中心划分为8个大区网络，这8个大区网共同构成了大区层，如图6-2所示。每个大区设两个大区出口，大区内部其他非出口节点分别与两个出口相连。省节点城市为天津、石家庄、呼和浩特、太原、兰州、西宁、乌鲁木齐、银川、昆明、贵阳、拉萨、杭州、福州、南昌、济南、合肥、郑州、长沙、南宁、海口、长春、哈尔滨，大区层主要提供大区内的信息交换，大区之间通信必须经过核心层。

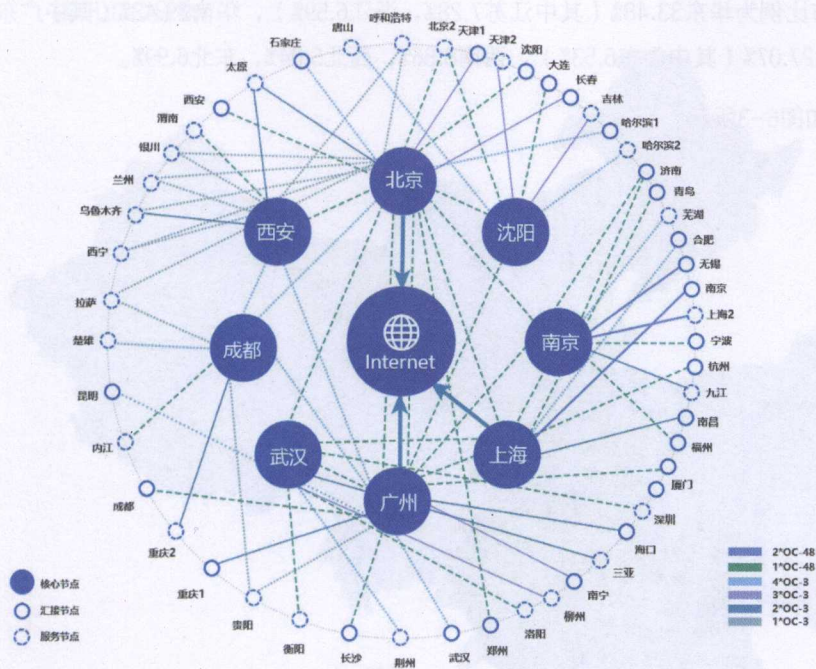


图6-2 大区层视图

IDC与CDN的区别很大

IDC与CDN本质上没有区别，可以认为IDC提供基础服务（带宽、机架、电等），CDN在IDC基础上提供增值服务。实际意义上是两个行业，IDC主要是主机托管和带宽租赁，而CDN主要是提供服务。如果把互联网看成平台、网络和终端用户这三个部分的话，CDN是基于网络之上的分发和优化传输技术，是对网络的细化和增值。从实际用途来看，IDC是更优质的资源，主要分布在核心城市，承载产品核心业务模块和基础平台，比如一幢房子好比一款产品，IDC具有“顶梁柱”的作用，支撑房子的基本价值，牢固的“顶梁柱”是房子可用和安全的基本前提。

IDC需要跟用户“走”

IDC存在的本质是“取悦”用户，IDC资源的投入是以运行应用提供给用户使用和好的体验为基本目的，基于千亿级别的真实用户访问数据得到省份、用户的分布属性信息如下。

- 1 用户最多的5个省份分别是广东、江苏、浙江、山东、河南，占总用户量的39.64%，分别分布在华南、华东、华北。
- 2 用户最少的20个省份占总用户量的26.37%，主要分布在偏远地区，而且非常散。

- 3 大区的用户分布比例为华东33.48%（其中江苏7.28%，浙江6.59%），华南29.42%（其中广东13.94%），华北27.07%（其中山东6.53%），西南8.86%，西北5.04%，东北6.99%。

全国所有用户分布如图6-3所示：

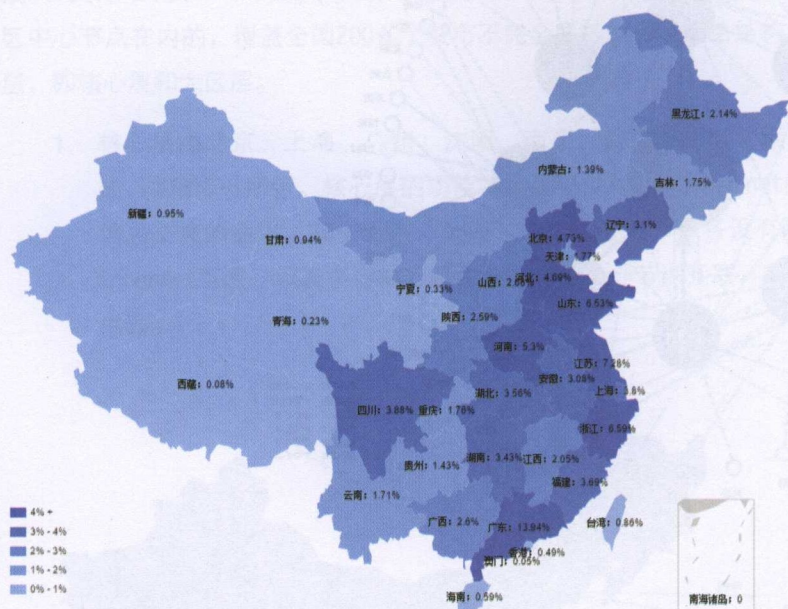


图6-3 用户分布视图

综上所述，IDC的分布按大区、运营商分布，并且离大区中大省用户越接近越好，同时需要考虑IDC的覆盖能力。可以参考BAT的做法，BAT各家核心产品特性各不相同，IDC的分布特性也是有差别的（非CDN部分）。腾讯主要是互动社区和游戏娱乐，早期电信、联通、小运营商主要集中在深圳，后来深圳到了无带宽可用的程度，陆续将业务大规模往西安（电信、联通）、上海（电信为主）、天津（联通为主）IDC迁移，目前全国五大区都有IDC和大容量专线互联，是IDC分布做得最好的，百度是搜索业务为主，主要集中在北京14个IDC，后面为了优化搜索体验，改善南方电信用户体验，陆续启用了杭州、南京等电信IDC，阿里以电商业务为主，主要运营商IDC集中在华东，以杭州为主。

腾讯IDC分布

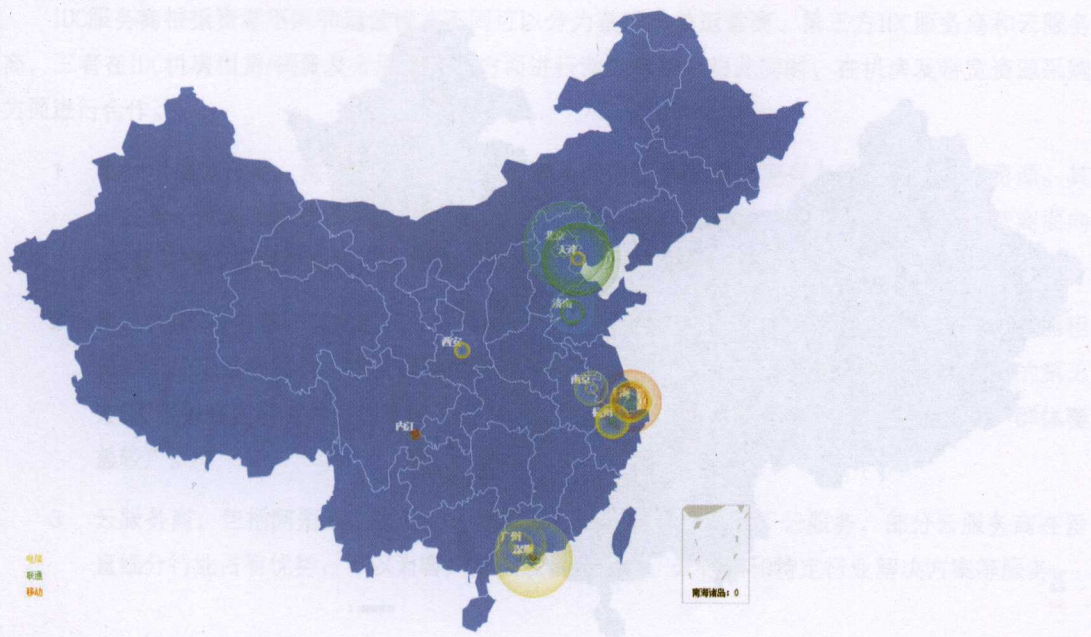


图6-4 腾讯IDC分布视图

百度IDC分布

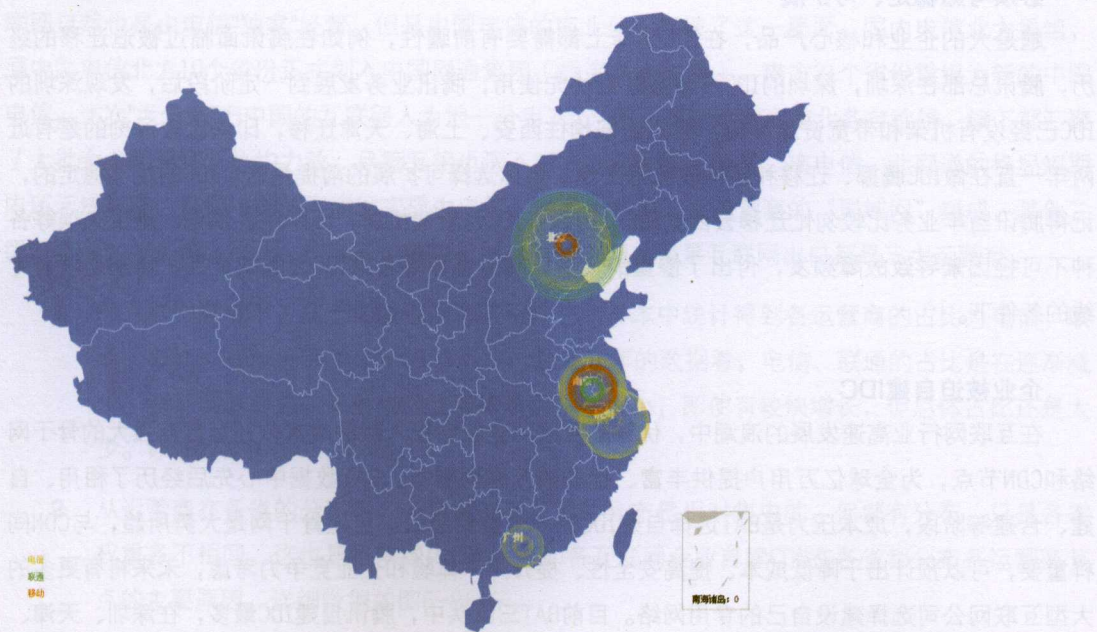


图6-5 百度IDC分布视图

阿里IDC分布

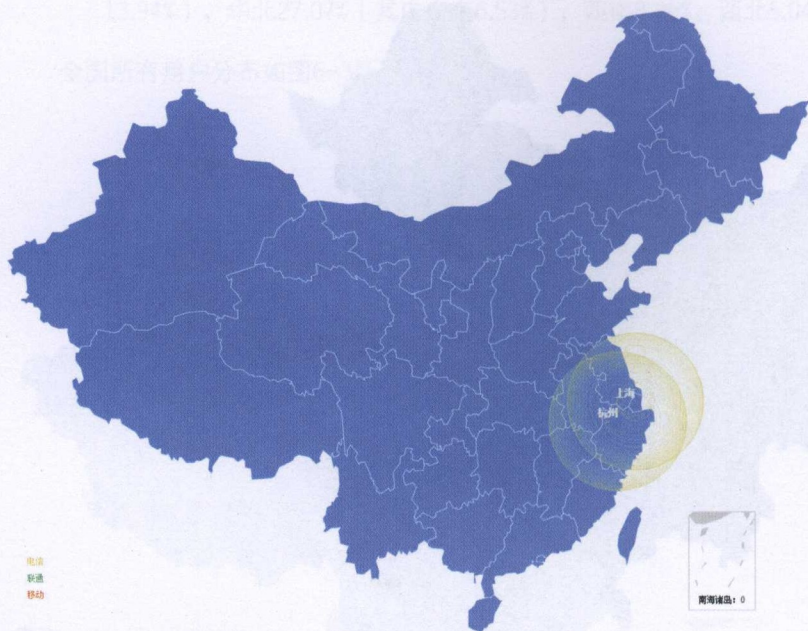


图6-6 阿里IDC分布视图

必须考虑稳定、可扩展

越是大的企业和核心产品，在IDC选择上都需要有前瞻性，例如在腾讯面临过被迫迁移的经历，腾讯总部在深圳，深圳的IDC资源被就近优先使用，腾讯业务发展到一定阶段后，发现深圳的IDC已经没有机架和带宽资源可用，需要调架构往西安、上海、天津迁移，印象比较深刻的是有近两年一直在做IDC腾挪、迁移和架构改造等工作。当然选择可扩展的前提是这个IDC的足够稳定的，记得腾讯当年业务比较匆忙迁移去西安IDC，因西安IDC为新建机房，因网络、供电、施工失误等各种不可控因素导致故障频发，付出了惨重的代价。同时也需要考虑IDC之间的交互，特别是没有专线的条件下。

企业被迫自建IDC

在互联网行业高速发展的浪潮中，优秀的互联网企业会投入巨额成本依托运营商强大的骨干网络和CDN节点，为全球亿万用户提供丰富、优质的互联网服务。BAT数据中心先后经历了租用、自建、合建等阶段，成本压力是BAT选择自建IDC的一个重要原因。自建骨干网是大势所趋，与CDN同样重要，可以预计出于降低成本、提高安全性、提升用户体验和企业竞争力考虑，未来将有更多的大型互联网公司选择建设自己的专用网络。目前BAT三巨头中，腾讯自建IDC最多，在深圳、天津、西安、上海等地进行了多期的IDC自建，百度、阿里紧随其后。

IDC服务商分类对比

IDC服务商根据资源不同和运营模式不同可以分为基础电信运营商、第三方IDC服务商和云服务商，三者在IDC机房租赁/销售及云服务产品方面进行激烈竞争，与此同时，在机房及带宽资源采购方面进行合作。

- 1 基础电信运营商，包括中国电信、中国联通和中国移动等，拥有大量的基础设施资源。其中在骨干网络带宽资源和互联网国际出口带宽方面具有垄断性优势。基础电信运营商面向IDC服务商、云服务商和行业客户提供互联网带宽资源及机房资源。
- 2 第三方IDC服务商，包括世纪互联、鹏博士和光环新网等。为客户提供主机托管、服务器租赁和机房运维等服务。按机房属性可划分为自有机房的第三方IDC服务商和租用机房的第三方IDC服务商。前者普遍拥有较强实力，主要面向大型客户群体，后者对中小型客户群体覆盖较广。
- 3 云服务商，包括阿里云、UCloud、盛大云和腾讯云等。专注于云服务，部分云服务商在垂直细分行业占有优势，可以为客户提供混合云主机、云存储和特定行业解决方案等服务。

6.2.2 ISP优化

其实中国的互联网原本是一张网，多年前中国电信几乎垄断着全国的固话网络，新兴的宽带互联网自然也是由电信“独家”经营。但是中国电信的南北分拆打破了这一局面。国内电信业大重组，原中国电信北方10个省份正式划入中国网通集团（后来并入联通），南方21个省份重组为新的中国电信，这次“大分家”把中国的互联网人为地一分为二。两大宽带运营商南北各自称雄，旗下都汇聚了大批企业和用户，势均力敌。尽管互相也深入对方地盘发展，但是，南电信、北网通的格局短期内还无法打破。目前中国的互联网实质由电信、联通、移动三大运营商的“局域网”组成，其他二级运营商比如长城、铁通、广电的天威等虽然自有接入网，但是互联网出口都是三大运营商。

- 1 PC互联网环境下，从千亿级别的真实用户访问样本中统计得到各运营商的占比为电信：联通：移动：其他=53%：35%：10%：2%，从近几年的数据看，电信、联通的占比是在逐渐减少，移动增速明显，其他小运营商因基数相对太小，即使有较快增长，但总体占比还是太少。详细数据如图6-7所示。
- 2 从运营商在各省的分布看，各运营商在各省的分布是相对集中的，但都有分布，只是各省权重各不相同，这也是目前国内CDN厂商和各互联网企业自建CDN在各省都分布各运营商节点的主要原因。详细数据如图6-8所示。

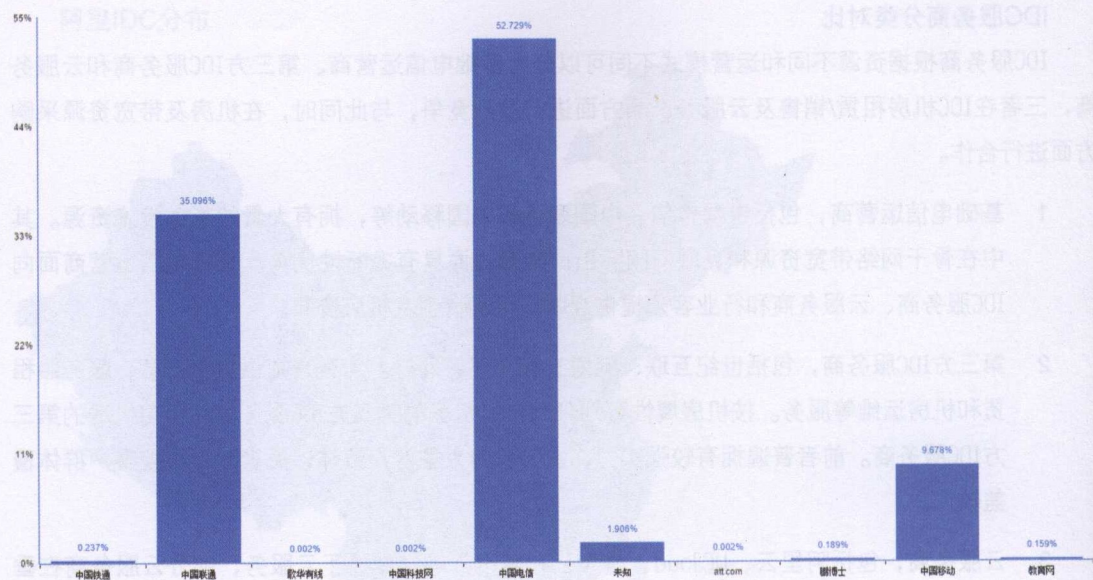


图6-7 运营商占比视图

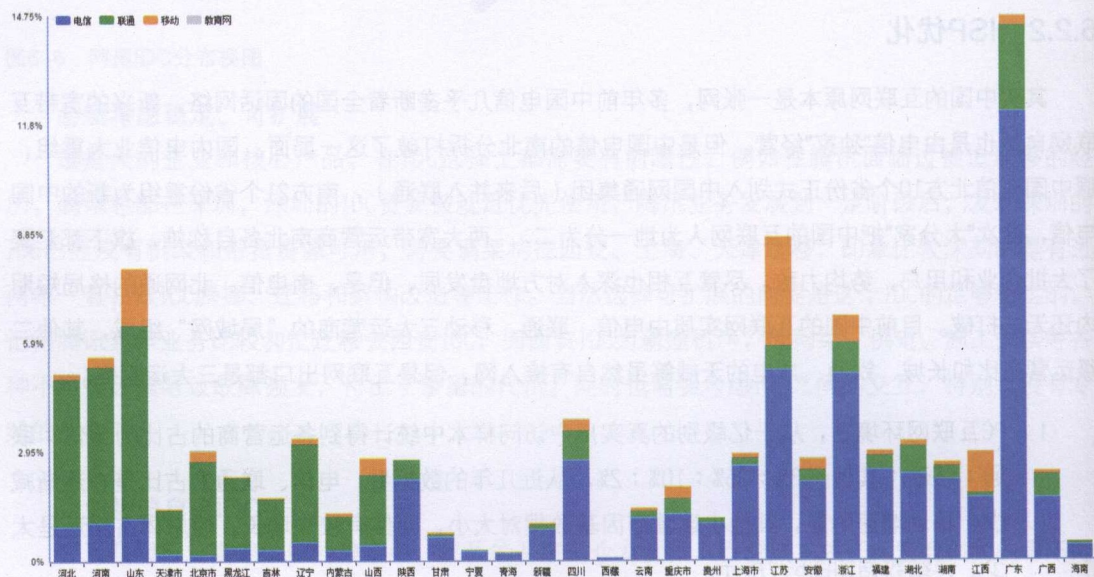


图6-8 省份运营商视图

3 运营商的地域性很明显, 74.3%的联通用户分布在华北、东北, 71.1%的电信用户分布在华东、华南。跨网访问速度慢2~3倍, 电信跨网访问联通最严重, 同运营商跨省解析是有递减效应的, 跨网与跨省影响是叠加的。详细数据如图6-9和图6-10所示。

川，天威视讯主要集中在深圳，歌华有线主要在北京。详细数据如图6-11和图6-12所示。

移动

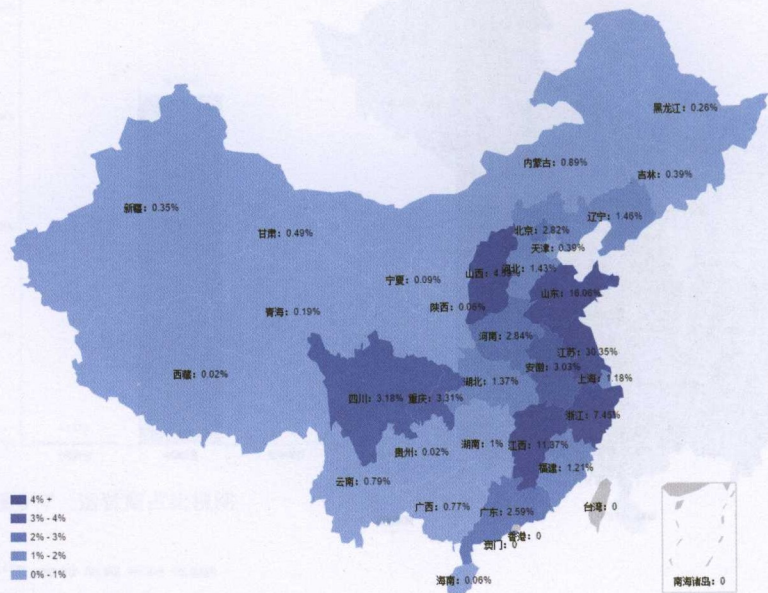


图6-11 移动用户省份占比视图

教育网

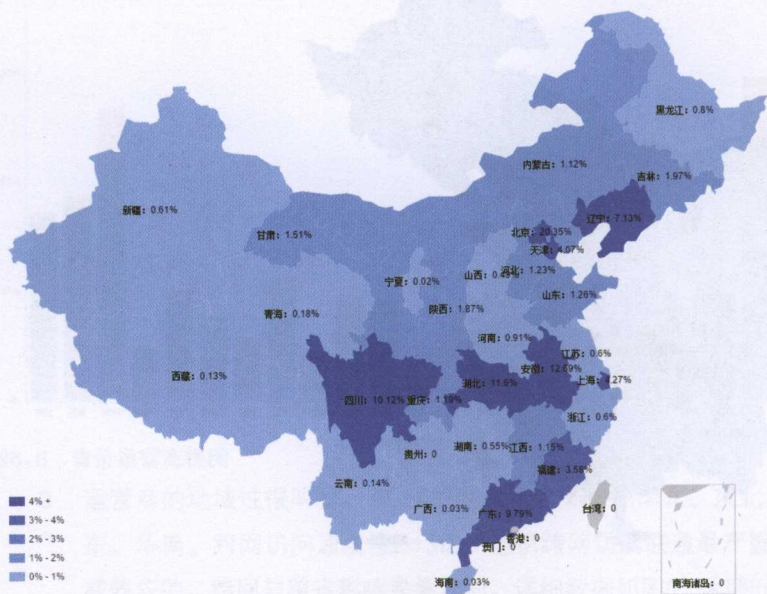


图6-12 教育网用户省份占比视图

6.2.3 CDN优化

CDN的全称是Content Delivery Network，即内容分发网络。其目的是通过在现有的Internet中增加一层新的网络架构，将网站的内容发布到最接近用户的边缘网络，使用户可以就近取得所访问的内容，解决 Internet网络拥挤的状况并提高互联网业务响应速度。从技术上全面解决由于网络带宽小、用户访问量大、网点分布不均等原因所造成的用户访问响应慢的问题，是改善用户业务体验的重要手段。从中国互联网的本质可以看到CDN、第三方带宽是中国特色互联网带来的行业性机会，随着用户接入提速、移动互联网产品爆发，还会继续高速增长。运营商之间互联链路长期高负荷、满负荷运行，扩容速度远远落后于用户接入带宽速度，尤其是固网光纤入户的换代升级和无线网络4G/5G升级，导致互联互通的扩容远远满足不了用户访问量的增长需求，CDN的分流更体现了补充的价值。

CDN厂商及加速类型

随着国内移动互联网、云计算、大数据等互联网新技术大踏步发展，CDN服务作为互联网内容的快递员，显得越发重要了。除了CDN技术的变革，CDN市场格局也发生了巨大转变，从最初的CDN专业服务提供商，到现在的电信运营商、互联网公司，都开始进军CDN行业，CDN已经成为互联网基础设施中不可或缺的重要组成部分。CDN服务商及优劣分析如表6-3所示。

表6-3 CDN服务商及优劣分析

CDN 服务商	优势	劣势
国内老牌CDN服务商，代表企业：蓝汛通讯、网宿科技、帝联科技、快网等	1. 专注核心业务发展，进入市场较早，易扩大经营规模，具有成熟的运营机制和较高的服务能力。 2. 服务的客户分布国内互联网主要行业，而且规模大，多源的服务能力和定制交付是竞争力。 3. 规模大、资源足，安全、性能等配套齐全，例如网宿储备带宽约12T，国内节点数近600个，海外节点60+	1. 受制于电信运营商带宽租用费用，带宽成本是刚性的，不能形成价格优势。由于电信运营商支撑系统的不灵活导致带宽资费设置不灵活，不能按需索取，导致CDN的价格居高不下，而且结算不灵活。 2. 多事业部、多节点组导致服务不一致。高负载、高复用，故障或突发流量影响所有业务
云CDN提供商，代表企业：阿里云、腾讯云、UPYUN	1. 自身CDN需求大及降低成本驱动自建，在自身大规模业务的实践后，不断在性能、成本上追求极致。 2. 互联网企业初期需要使用第三方CDN分担业务快速增长的压力，同时自建CDN时，取长补短，更好地契合自身业务。 3. 规模效应明显，与运营商议价空间大，阿里CDN服务带宽6T，节点数近500个，海外节点30+，腾讯CDN服务带宽10T，节点400+	1. 初期投资较大，且增加运营和研发成本，初期缺少CDN研发、选点和运营经验。 2. 非核心业务，易分散对主营业务的精力，对外支撑能力有限，与对内服务产品线不同，对外服务企业 数量是对内的数倍，而且沟通与客户需求更多源化。 3. 商业CDN要求更高，需要更全面的CDN产品和配套的线上、线下服务支撑能力

续表

CDN 服务商	优势	劣势
代理CDN厂商，代表企业：金山云（部分）、UCloud、七牛等	1. 可以减少CDN初期带宽、服务器投入，并且将服务能力转嫁给专业CDN服务商。同时与多家CDN服务商使用，节点选择性更强，性能更容易保障，同时也增加了议价权。 2. 付费体系和网上服务较清晰，付费模式灵活、方便。传统 CDN 服务厂商价格体系不公开，无法实现网上支付	1. 作为中间商，不能直接控制风险，即使出现故障或网络抖动也是被动响应，需要CDN服务商解决，而且影响所有用户。 2. 非主营业务，研发和人力投入有限，服务支撑能力较弱
电信运营商，代表企业：中国电信、中国联通	1. 拥有巨大的带宽和网络优势，其带宽成本是弹性的，可以配置最优的CDN服务网络资本雄厚，有实力建设规模庞大的CDN网络，拥有品牌优势，与众多互联网公司关系密切，能够引导互联网公司使用CDN服务。 2. 已建成页面、流媒体等专用CDN网络在此基础上进行升级和改造，可快速推出多种CDN业务，服务带宽初成规模，达到3~4T左右	1. CDN行业属于技术驱动型产业，电信运营商的CDN技术储备较薄弱。 2. 互联互通问题导致运营商只能在自己的网络上做CDN业务，跨网络运营的问题较难解决。 3. CDN不是电信运营商的主营业务，缺乏运营经验和服务能力
国外CDN服务商，代表企业：Akamai、Limelight Networks等	1. 资金雄厚，具备技术、运营和服务实力，利于更好的提升业务性能。 2. 在国内开始建设节点，将国际先进的技术引进中国，利于行业竞争和发展	1. 与国内一些IDC和ISP采取代理合作的方式来销售自己的服务，难以大规模开展业务。 2. 非正规竞争，存在安全隐患，将域名解析权掌控在国外运营商的手里

CDN节点及调度决定性能

在中国，CDN由网络节点和调度决定了主要加速效果，即使更高性能的硬件和更前沿的内容优化、动态加速和TCP加速，也只有在最大可能CDN节点分布的基础上，才能达到最佳效果。首先有好的CDN节点，但国内CDN节点达2800个之多，而且还不断增加，这些节点性能和稳定性参差不齐，要从这些节点中找出优秀的需要持续测试和较长的时间来验证。选择做节点还是第一步，能否将最适合访问的用户解析到这些节点还有较多的困难，后面章节会详细介绍。

- 1
- CDN选点是第一步，无论老牌CDN厂商，还是BAT自建CDN，全国无非也只有300~500个节点可选。可以肯定这些节点都不能绝对保障稳定性，特别是第三方CDN都是共用的，不确定的突发或区域流量不均衡是常态。通常按运营商，按优选级从一线>二线>三线城市的节点开始挑选，每家CDN服务商能提供几组资源组，每组节点都是批量配置，不能单选，所以可选择的空间也是有限的。自建CDN节点可控性相对要好，按批上线节点，不断淘汰质量差、不稳定的节点，留下来的再精选，直到稳定、平衡，当然首先要能持续监测这些节点的质量。图6-13为淘宝CDN节点分布，以便帮助大家选择。

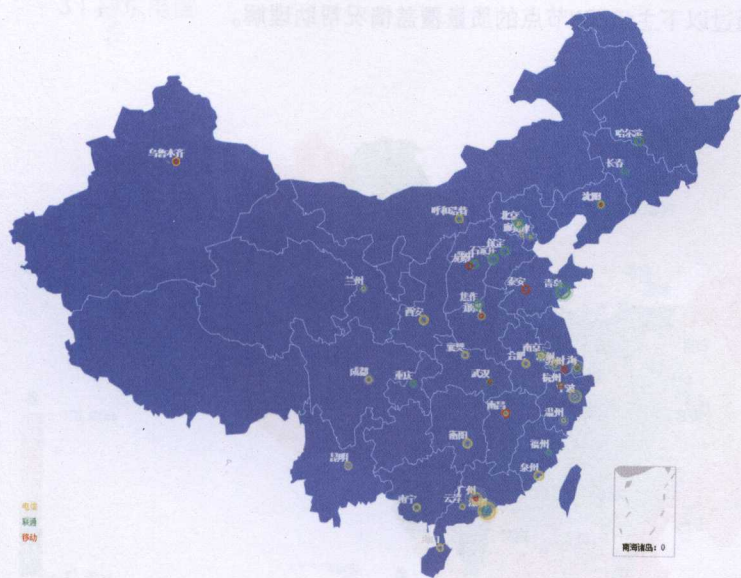


图6-13 CDN节点分布分析

- 2 CDN节点解析与选点一样重要，如果有优质的CDN节点，最合适的用户没有解析到这些节点，其实意义不大。使用过第三方CDN的人应该都知道“换节点”这一潜规则，CDN服务商从测试阶段、签约前期、签约中后期给出的节点是不一样的，当然也有节点网络、硬件故障等不能使用，需要时刻监测CDN节点的解析情况，及时对不合理的解析做优化。如何监测解析，在之前监测章节中有详细介绍。图6-14为淘宝CDN节点解析情况，以便帮助大家选择。

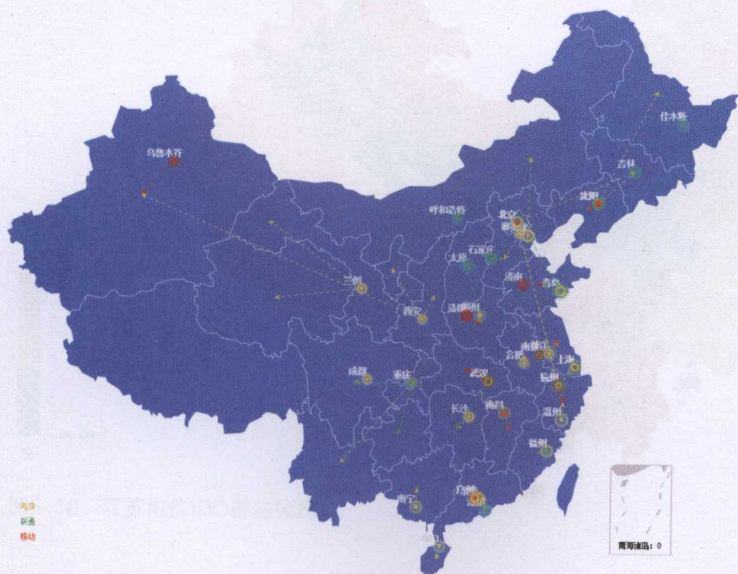
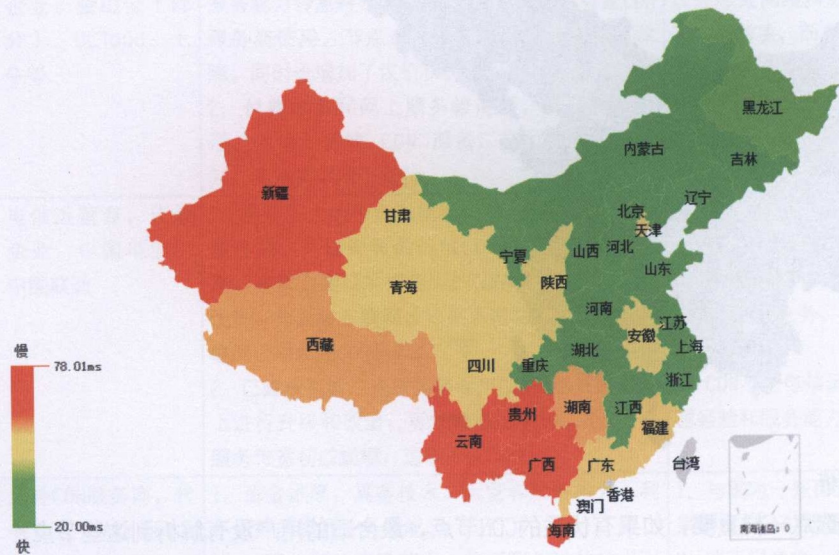


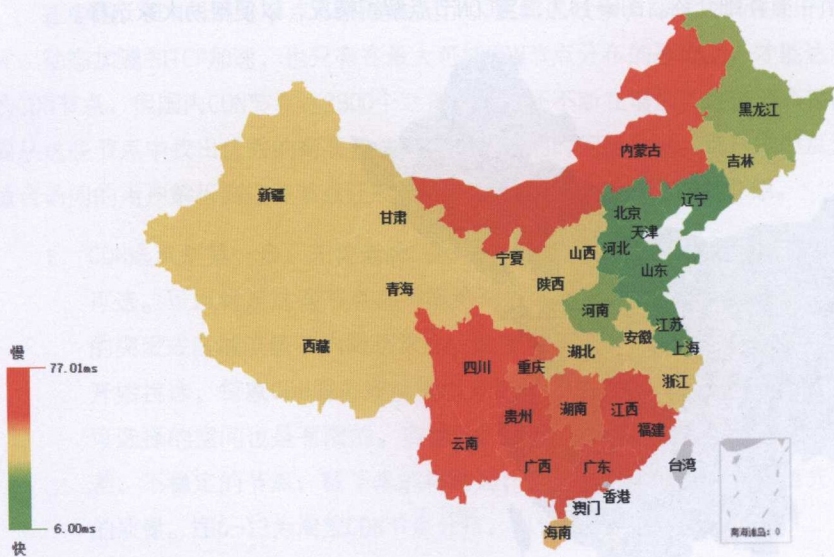
图6-14 CDN解析分析视图

3 CDN节点及解析，可以通过以下主流CDN节点的质量覆盖情况帮助理解。

1) 北京联通



2) 天津联通



3) 广东电信

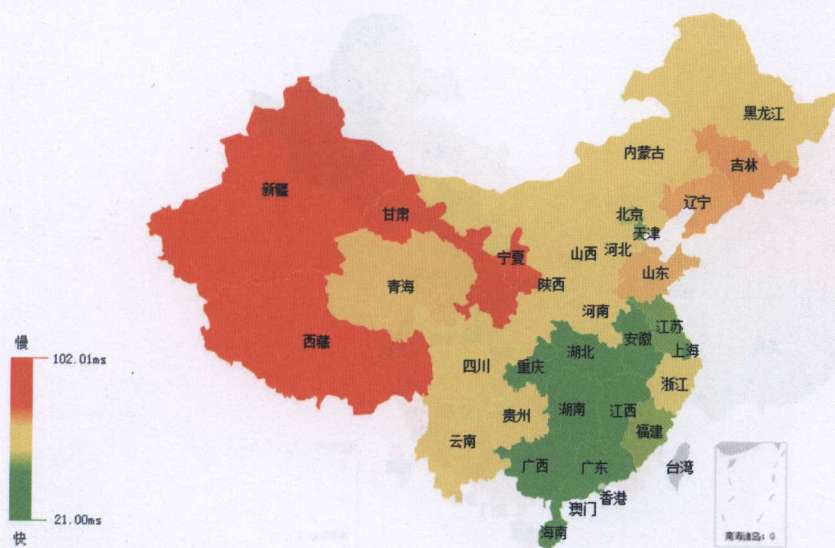


图6-17 广东电信IDC覆盖视图

4) 江苏电信

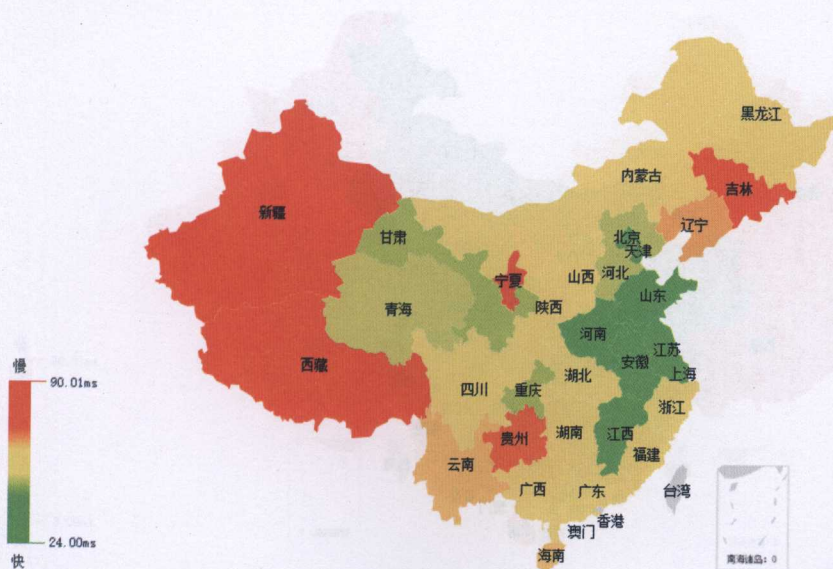


图6-18 江苏电信IDC覆盖视图

5) 吉林电信

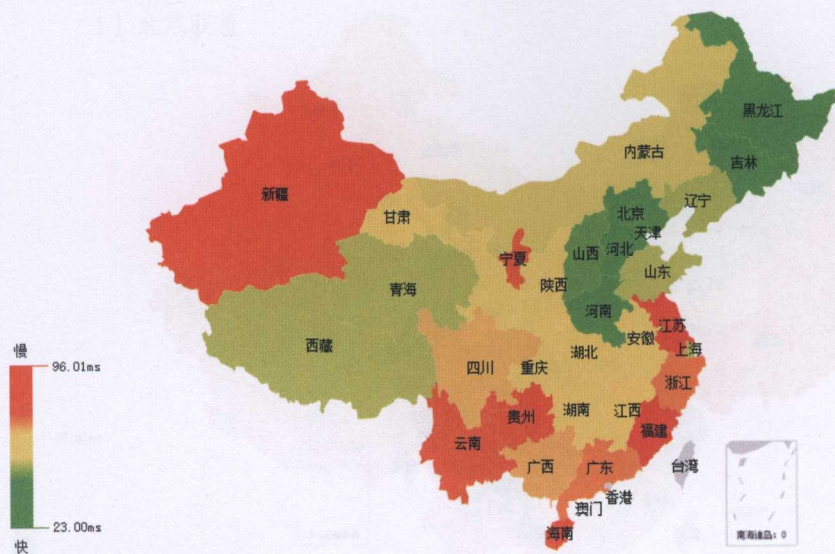


图6-19 吉林电信IDC覆盖视图

6) 辽宁电信

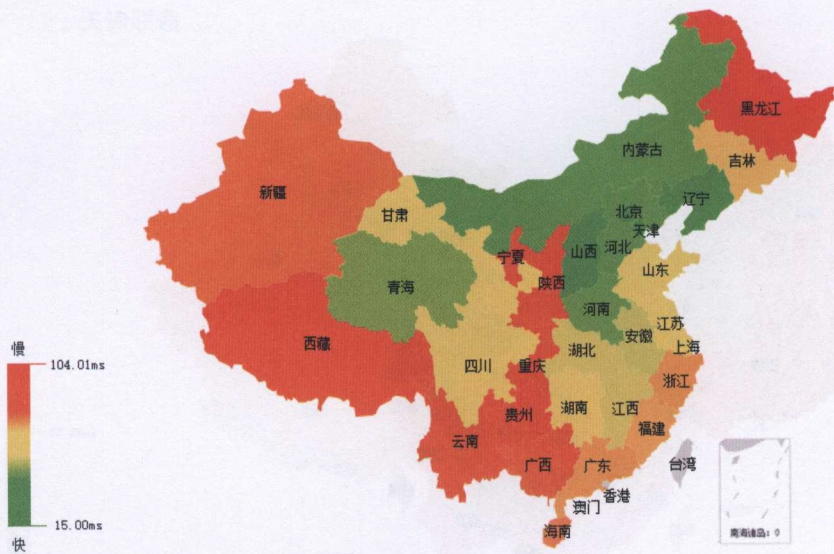


图6-20 辽宁电信IDC覆盖视图

7) 湖北移动

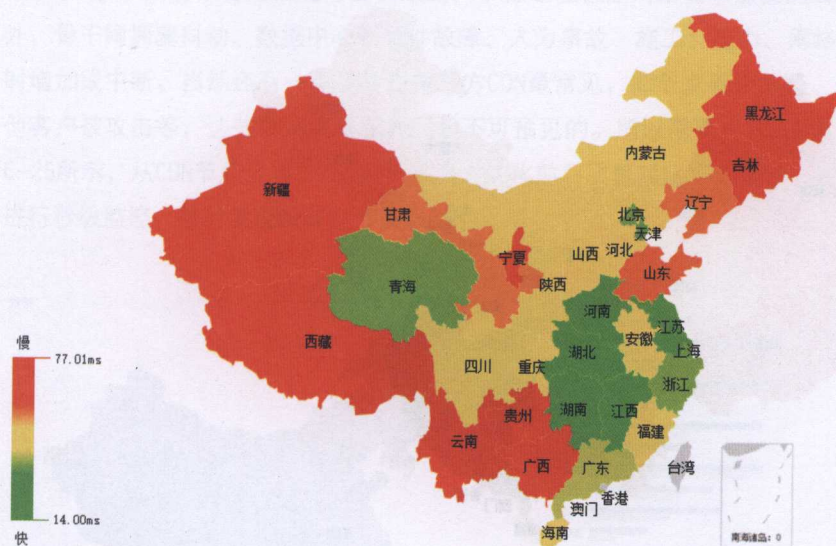


图6-21 湖北移动IDC覆盖视图

8) 上海电信

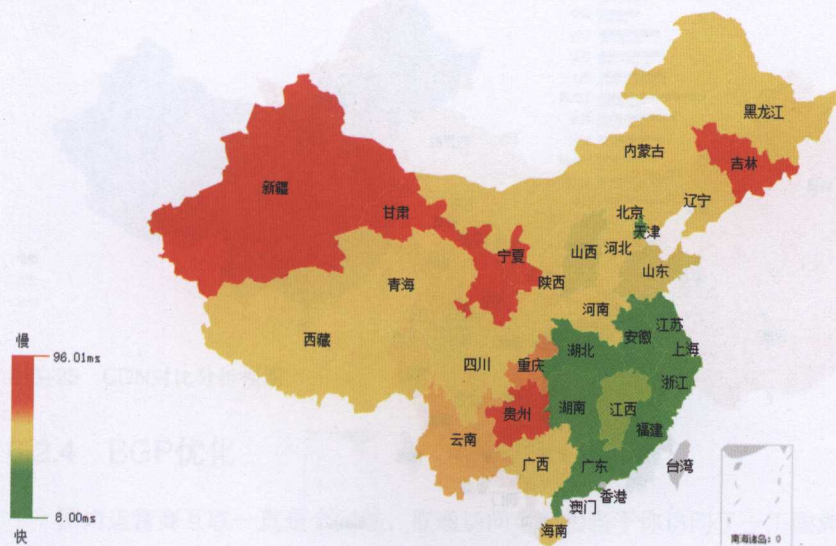


图6-22 上海电信IDC覆盖视图

9) 浙江电信

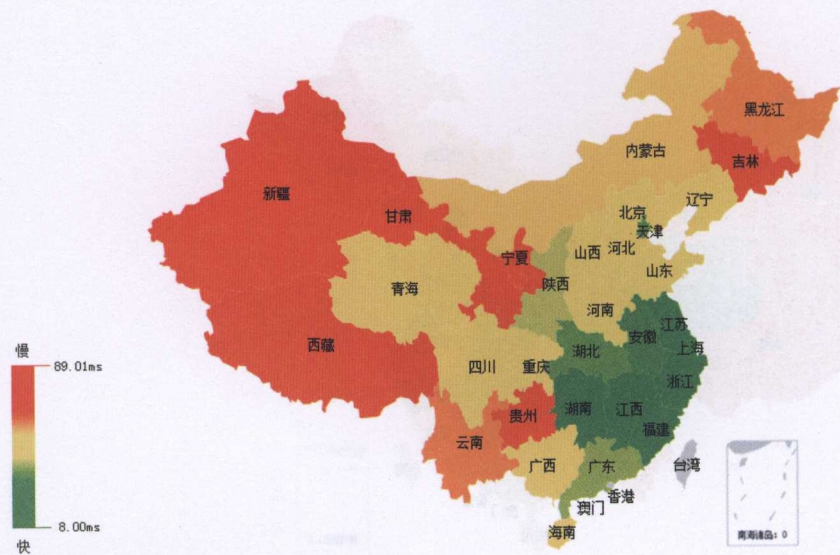


图6-23 浙江电信IDC覆盖视图

10) 四川电信

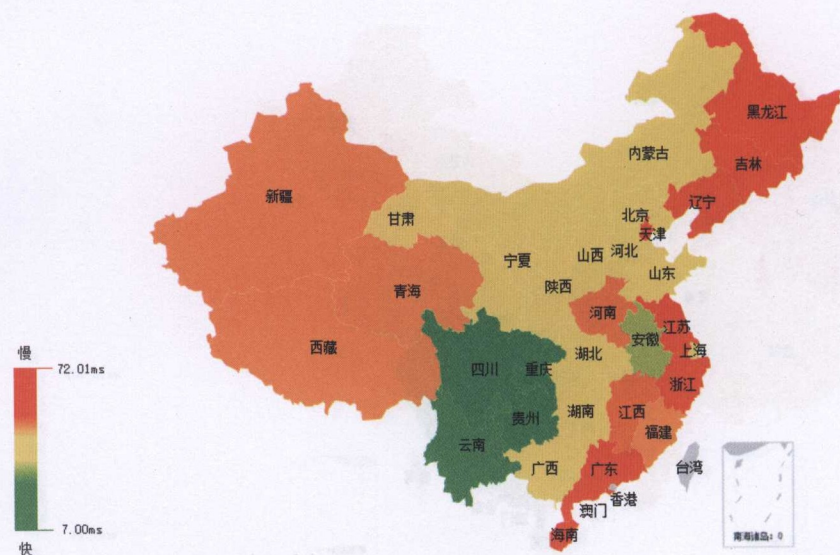


图6-24 四川电信IDC覆盖视图

重视监测对CDN的作用

CDN节点具有不稳定性是与生俱来的，不稳定性也是网络资源重要的属性。除本身的性能优劣外，骨干网拥塞抖动、数据中心软硬件故障、人为事故、施工、电力、网络攻击等都会导致网络延时增加或中断，当然还有一类故障在第三方CDN最常见，即节点硬件故障、负载超载、节点上的其他客户被攻击等，这些影响频率和时间是不可预见的。所以需要建立CDN实时质量监测体系，如图6-25所示，从CDN节点分布、CDN解析策略、优化前后质量对比（分省份、城市、可用率三个维度）进行秒级监控，随时发现运营商节点级别的异常。

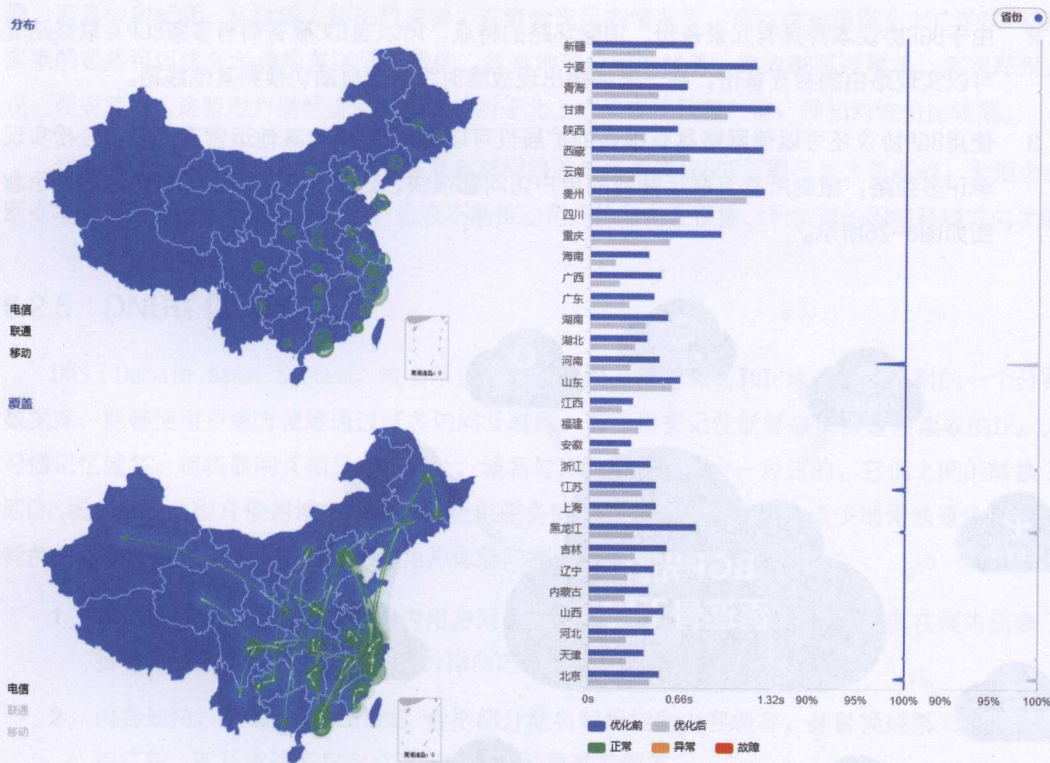


图6-25 CDN对比分析视图

6.2.4 BGP优化

国内运营商互联一直是个问题，联通访问电信相当于你访问了一下国外网站一样，对网络访问有很大影响。BGP（边界网关协议）协议主要用于互联网AS（自治系统）之间的互联，BGP的最主要功能在于控制路由的传播和选择最好的路由。中国网通与中国电信都具有AS号（自治系统号），全国各大网络运营商多数都是通过BGP协议与自身的AS号来互联的。使用此方案来实现双线路需要在

CNNIC（中国互联网络信息中心）申请IDC自己的IP地址段和AS号，然后通过BGP协议将此段IP地址广播到网通、电信等其他的网络运营商，使用BGP协议互联后网通与电信的所有骨干路由设备将会判断到IDC机房IP段的最佳路由，以确保多运营商之间的高速访问。

BGP解决中间一公里的问题

- 1 服务器只需要设置一个IP地址。最佳访问路由是由网络上的骨干路由器根据路由跳数与其他技术指标来确定的，不会占用服务器的任何系统资源。服务器的上行路由与下行路由都能选择最优的路径，所以能真正实现高速的单IP双线访问。
- 2 由于BGP协议本身具有冗余备份、消除环路的特点，所以当IDC服务商有多条BGP互联线路时可以实现路由的相互备份，在一条线路出现故障时路由会自动切换到其他线路。
- 3 使用BGP协议还可以使网络具有很强的扩展性可以将IDC网络与其他运营商互联，轻松实现单IP多线路，做到所有互联运营商的用户访问都很快，这个是双IP双线无法比拟的，示意图如图6-26所示。



图6-26 BGP示意图

BGP与IDC、CDN的区别

BGP属于IDC，都是资源，而CDN是网络加速服务，CDN解决最后一公里的问题。简单理解BGP是让多种线路的用户通过单IP访问你的网站或应用，即BGP是多线路访问服务器，而CDN是多网络节点让用户更快地访问你的网站，CDN侧重是提供更多节点来加速访问过程。选择BGP通常是部署系统最

关键的模块，注重多运营商用户访问及系统构架本身动态数据交互的效率，而选择CDN主要解决用户就近访问及通过多CDN节点缓存内容分流突发访问带来的对系统的冲击及成本优化考虑。

BGP的短板及规避措施

BGP机房的缺陷在于多线汇聚的接入线路，本身就是外地二级线路引入的，比如南方的BGP引入的南方联通线路本身对北方联通就覆盖不好，如果从北方联通或者北方BGP拉专线进来做汇聚，成本很高，而且带宽有限，所以BGP也具有地域属性，需要与各大区IDC配合并分布式部署。另外，最好的BGP机房主要分布在北上广地区，这三个地区是我国的骨干网络地区，网络速度、稳定性可以保障，而且机房配套、运营能力都比较完善，在资金充足的情况下，可以优先选择北上广的BGP机房，实惠的选择可以优先考虑华东江浙一带的，具有地理位置的优势，南北都可以覆盖，其次郑州、武汉、西安等地，这些地方虽然说网络状况相对于北上广来说要稍差一些，但相对性价比比较高。

目前大型互联网企业、主流云服务商自建网络并与多方实现BGP互联已是大势所趋。大型企业不断在增强自有业务竞争能力的同时，也在不断推动用户体验向更便捷、更快速的网络环境方向发展。

6.2.5 DNS优化

DNS (Domain Name System, 域名系统)，互联网上作为域名和IP地址相互映射的一个分布式数据库，能够使用户更方便地通过域名访问互联网，而不用去记住能够被机器直接读取的IP。人们习惯记忆域名，但机器间互相只认IP地址，域名与IP地址之间是一一对应的，它们之间的转换工作即DNS域名解析。但凡使用域名来给用户提供服务的互联网企业，都或多或少地无法避免在有中国特色的互联网运营商环境中出现各种用户体验问题，主要有以下三类。

- 1 本地缓存，各运营商确保网内用户网内访问，同时减少跨网结算，运营商在网内搭建了内容缓存服务器，通过把域名强行指向内容缓存服务器。
- 2 内容劫持，有部分 LocalDNS 会把部分域名解析指向内容缓存，并替换成第三方广告联盟的广告。PC环境的客户端也存在大量的恶意劫持现象。
- 3 解析转发，运营商的 LocalDNS 还存在解析转发的现象。解析转发是指运营商自身不进行域名递归解析，而是把域名解析请求转发到其他运营商的递归 DNS 上的行为。解析请求的来源 IP 成了其他运营商的 IP，最终导致跨网而使用户访问变慢。

综上所述，DNS优化是非常有必要的，DNS优化主要有以下几个途径。

TDO

TDO (Traffic Distribution Optimizer)，可以理解是一种用户定向策略，DNS以TDO为依据，

将用户解析到对应的运营商及服务器，保证使用最佳的服务器服务最近的用户，从而确保访问速度，正确的定向可以提升用户访问速度，反之将适得其反。百度旧版TD0因历史原因，导致存在7%的跨网访问，通过使用Apnic、MaxMind等数据修正跨网解析和未知Local DNS后，速度提升明显，而且是所有产品线受益。

TD0解析数据分为两大部分：IP库以及IP-区域映射库。

- 1 IP库，即IP（IPv4/IPv6）地址段与地域信息的映射关系，地域信息包括：运营商、国家、省份、城市、县、区等。也可以结合了IP服务提供商以及自主维护的IP信息数据。
- 2 IP段-区域映射库，就是将IP段信息按照一定层次，组织成不同的区域，相邻层次区域之间有着细化和泛化的关系。例如，用户IP的信息为：中国-江苏-南京-电信，则这个IP可以对应的区域，如图6-27所示：

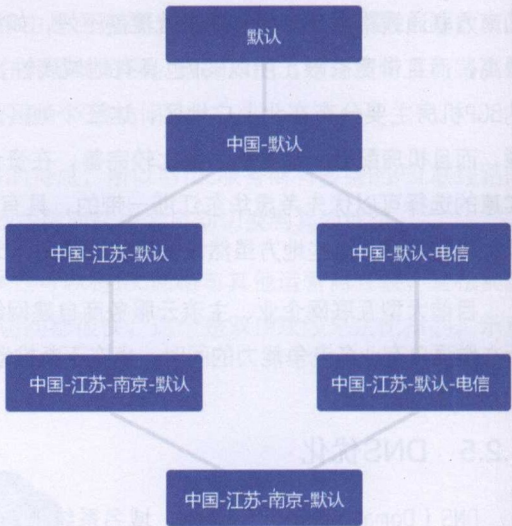


图6-27 TD0区域层次示意图

对于IP库中的每一个IP段，都可以将其对应到多个区域中，最终形成一个IP段-区域的映射矩阵，这样就生成了TD0解析数据。TD0解析数据，即IP库和IP段-区域映射库这两部分的数据，就为域名解析层的解析策略提供了数据基础。不过，由于地区IP段会不断调整，以及新入网用户的原因，TD0数据需要进行持续的人力和技术投入，以达到较高的准确性。

HttpDNS

HttpDNS是使用HTTP协议向DNS服务器的80端口进行请求，代替传统的DNS协议向DNS服务器的53端口进行请求，绕开了运营商的Local DNS，从而避免了使用运营商Local DNS造成的劫持和跨网问题。HttpDNS是为移动客户端量身定做的基于Http协议和域名解析的流量调度解决方案，主要解决Local DNS解析异常以及流量调度不准。

- 1 根治域名解析异常，由于绕过了运营商的Local DNS，用户解析域名的请求通过Http协议直接透传到服务器IP上，用户在客户端的域名解析请求将不会遭受到域名解析异常的困扰。
- 2 调度精准，HttpDNS能直接获取到用户IP，通过结合IP地址库以及测速系统，可以保证将用户引导的访问最快的IDC节点上。

- 3 实现成本低廉，接入HttpDNS的业务仅需要对客户端接入层做少量改动，无需用户手机进行root或越狱。而且由于Http协议请求构造非常简单，兼容各版本的移动操作系统更不成问题。另外HttpDNS的后端配置完全复用现有权威DNS配置，管理成本也非常低。
- 4 扩展性强，HttpDNS提供可靠的域名解析服务，业务可将自有调度逻辑与HttpDNS返回结果结合，实现更精细化的流量调度。比如指定版本的客户端连接请求的IP地址，指定网络类型的用户连接指定的IP地址等。

HttpDNS要尽可能部署BGP环境，保证各运营商的用户能够快速访问到HttpDNS服务。同时HttpDNS也需要在多个数据中心进行了部署，任意一个节点发生故障时均能无缝切换到备份节点，保证用户解析正常。

GSLB

GSLB (Global Server Load Balance, 全局负载均衡)，实现在广域网（包括互联网）上不同地域的服务器间的流量调配，保证使用最佳的服务器服务离自己最近的客户，从而确保访问质量。GSLB能通过判断服务器的负载，包括CPU占用、带宽占用等数据，决定服务器的可用性，同时能判断访问者与服务器间的链路状况，选择链路状况最好的服务器。因此GSLB是对服务器和链路进行综合判断来决定由哪个地点的服务器来提供服务，实现异地服务器群服务质量的保证。通常采用静态地域拓扑+动态调整策略实现全局负载均衡，架构如图6-28所示。

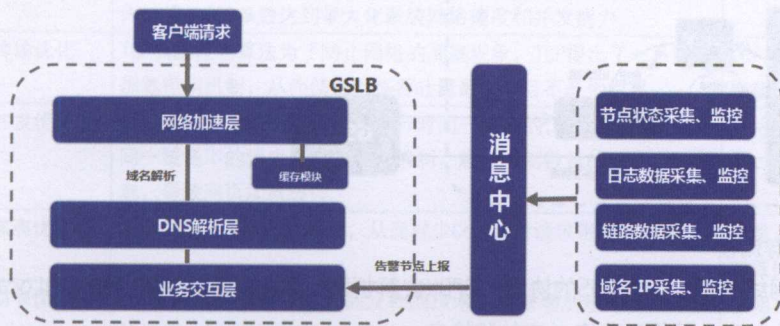


图6-28 GSLB技术框架

- 1 网络加速、缓存模块。主要为提高域名解析的速度。网络加速主要对网络内核进行优化，减少网络传输对DNS性能的影响；对于同一地区、ISP下的域名请求，通过缓存模块能够减少解析流程，从缓存数据中直接返回解析结果。
- 2 DNS解析层。使用开源的BIND-DLZ作为GSLB的DNS解析引擎，充分利用该软件的运行稳定、解析性能高、便于定制化开发、快速迭代等特性，解析策略主要有3种，如表6-4所示。

- 3 业务交互层。负责与DNS解析层的交互，负责接收消息中心的节点告警信息，并根据静态地域的拓扑配置，对BIND的数据记录进行CRUD操作。
- 4 消息中心、采集监控系统。这几个模块都是基于用户、应用监测数据，将OS监控、可用性监控等告警数据经过消息中心的过滤、路由，将节点的告警信息发送给GSLB进行调度。

表6-4 解析策略对比表

对比维度	基于DNS	基于HTTP重定向	基于IP路由
请求压力	由于有local DNS和用户端DNS缓存，请求压力不集中	接收所有请求，请求压力大，容易变成性能瓶颈	借助IP网络设备完成负载均衡，没有单点性能瓶颈
准确性	依据local DNS IP，定位用户不准确（可通过EDNS解决）	直接接收用户IP，定位准确	基于负载的调度，分发策略有限
扩展性/通用性	扩展性、通用性较好	只支持HTTP协议，其他应用协议需要定制开发	通用性好，但所有的服务器必须支持IP Tunneling 协议
使用场景	Web加速（静态资源）	流媒体、文件下载	无商用案例

GSLB的网络加速使用DPDK实现，能够跨过内核，直接从网卡收集数据以提高数据流在服务器中的传输速度。GSLB在缓存的实现过程中未使用基于socket通信的缓存，而是使用基于共享内存的Hash存储作为缓存，以提高数据的读、写速度。架构如图6-29所示。

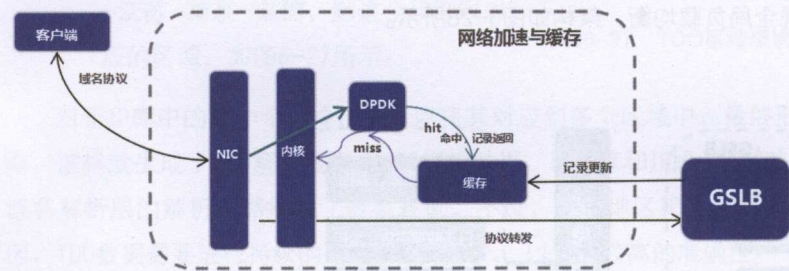


图6-29 GSLB网络加速与缓存流程

- 1 DPDK捕获到NIC的网络数据，对于DNS的协议，DPDK会解析DNS请求协议以获取Client的Local DNS的IP，查询IP信息库，根据IP获取对应的IP信息。
- 2 根据view信息（IP信息+域名唯一确定）查找缓存，如果命中则直接组织成DNS应答协议返回给客户端；未命中的则返回给NIC，将DNS请求协议按正常流程转发给GSLB进行后续处理。

6.3 系统优化

这里的系统是指单台硬件服务器或服务器集群，也包括服务器上运行的操作系统及应用，系统

层的优化也将针对以上环节及策略进行。除开网络层，用户访问互联网产品所得到的所有内容，都要从系统层生成并返回给用户，所以系统层的各环节也决定了速度的快慢，影响的范畴与网络层一致，是全范围的影响，结合个人的经验，将主要的系统优化实践汇总如表6—5所示：

表6-5 系统速度优化建议表

系统优化项	优化说明	备注
压缩优化	Gzip是目前最流行也是最有效的文本压缩方式。Gzip的思想就是把文件先在服务器端进行压缩，然后再传输。这样可以显著减少文件传输的大小。传输完毕后浏览器会重新对压缩过的内容进行解压缩并执行。目前的主流浏览器都能良好地支持 Gzip。而且Gzip的压缩比例非常大，一般压缩率为85%，就是说服务器端100K的文本可以压缩到25K左右再发送到客户端	Gzip对基础页和文本文件的加载速度有质的飞跃，甚至直接影响首屏速度
缓存优化	无论终端、前端、后端，都需要将最热的数据cache在第一层（通过淘汰策略去除冷数据），并要设计多级缓存的，对于冷数据或搜索数据的索引，更新不会很频繁，一般放二级缓存。多级缓存策略要体现在多ISP、IDC特性，特别是静态应用，要尽最大可能接近用户	互联网应用，缓存是永恒不变的话题，也是必修课；静态最大化缓存，甚至有策略的做缓存是必要的，简单有效，但在百度很多产品中仍存在很多不做缓存的情况
分离优化	产品在上线初期多数是混用的，初期访问量少的情况下，影响有限，但上规模后，非常有必要将静态进行剥离，使用最优静态架构、服务器和OS配置(退役旧设备)，动态做域名数量收敛	内容拆分是硬件选型、CDN加速和Web Server配置优化的前提
内核优化	内核是操作系统的核心，它使操作系统与安装的任何一种软件都可以和计算机硬件进行通信，根据应用场景，通过修改Linux的内核相关TCP参数达到最大化系统网络速度和并发能力	需要平衡CPU资源、Web Server配置，最好做到因地制宜
传输优化	TCP拥塞控制算法为了防止网络的拥塞现象，TCP提出了一系列的拥塞控制机制，从而使得网络吞吐量最大化且不产生拥塞	TCP拥塞控制是从系统层提升内容传输效率的重要优化措施
并发优化	浏览器的并发请求数目限制是针对同一域名的，即同一时间针对同一域名下的请求有一定数量限制，超过限制数目的请求会被阻塞，导致网页加载缓慢	域名拆分的前提是网页元素足够多，元素少可以不必考虑
隔离优化	主要隔离不必要的Cookie，从而减少Cookie对请求带来的影响	通常通过域名拆分提升浏览器并行加载优化的同时也实现了Cookie隔离
网卡优化	Linux网络协议栈中一系列互补的技术，用来增加多处理器系统的并行性和改善性能，以便充分利用CPU多核和硬件网卡等自身性能，达到并行、并发处理的目的	多用于高并发的业务接入层服务，充分发挥硬件潜力并提升性能。随硬件不断迭代，这个优化方向会越来越重要
硬件优化	Web服务性能由应用类型来决定的，如果是静态的，系统瓶颈是网络子系统和内存。如果动态应用或逻辑服务、数据库服务主要进行密集计算，系统瓶颈主要是内存、CPU、磁盘子系统和网络子系统；缓存服务、存储服务也要针对业务优先级选择不同的存储介质	精益求精最好，让应用层的性能达到极致

6.3.1 压缩优化

Gzip是GUNzip的缩写，是使用无损压缩算法的一种，最早是用于Unix系统的文件压缩，凭借着良好的压缩效率，现在已经成为Web上使用最为普遍的数据压缩格式。Gzip格式是一种很普遍的压缩技术，几乎所有的浏览器都有解压Gzip格式的能力，而且效果是非常明显的，大约可以减少70%以上的文件大小。这取决于文件中的内容。Web服务器将要发送的网页经过压缩后，不但提高了并发量和减少网页传输时间，加快网页加载和浏览，而且还会节省40%~60%的带宽成本。

压缩是如何工作的

客户端向Web服务器端发出了请求后，通常情况下服务器端会将页面文件和其他资源，返回到客户端，客户端加载后渲染呈现，这种情况文件一般都比较大，如果开启Gzip，那么服务器端响应后，会将HTML、JS、CSS等文本文件或者其他文件通过高压缩算法将其压缩，然后传输到客户端，由客户端的浏览器负责解压缩与呈现。

客户端请求报文中包含Accept-Encoding表示客户端能识别的压缩方法，如果客户端请求报文没有包含Accept-Encoding首部，服务器就会假设客户端能够接受任何编码格式，服务器响应报文中包含Content-Encoding表示采用的压缩方法。

- 1 Web服务器接收到浏览器的HTTP请求后，检查浏览器是否支持HTTP压缩（Accept-Encoding信息）。
- 2 如果浏览器支持HTTP压缩，Web服务器检查请求文件的后缀名。
- 3 如果请求文件是HTML、CSS等静态文件，Web服务器到压缩缓冲目录中检查是否已经存在请求文件的最新压缩文件。
- 4 如果请求文件的压缩文件不存在，Web服务器向浏览器返回未压缩的请求文件，并在压缩缓冲目录中存放请求文件的压缩文件。
- 5 如果请求文件的最新压缩文件已经存在，则直接返回请求文件的压缩文件。
- 6 如果请求文件是动态文件，Web服务器动态压缩内容并返回浏览器，压缩内容不存放到压缩缓存目录中。

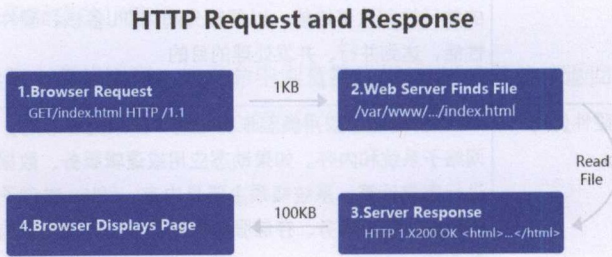


图6-30 未使用Gzip示意图

未使用Gzip如图6-30所示。

开启使用Gzip如图6-31所示。

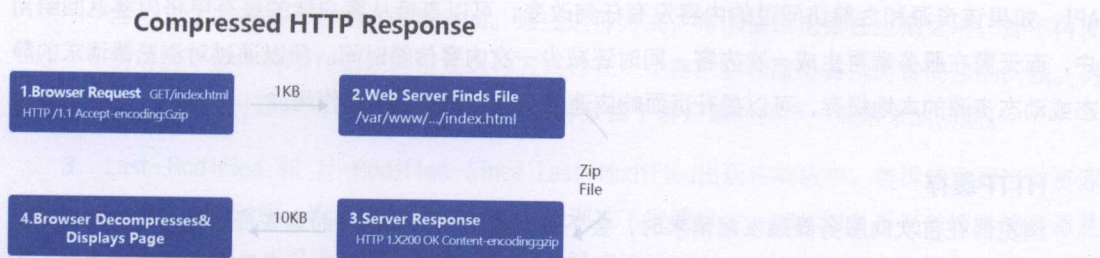


图6-31 使用Gzip请求示意图

然而，一个统计表明，大约有15%的客户端请求是没有Accept-encoding请求的，因为客户端的一些Web代理和PC安全软件会移除浏览器发出的Accept-encoding，原因在于监听未经压缩的响应会占用更少的CPU资源，但却无疑增加了网络传输的时间。

压缩的优缺点

总体来说，使用压缩还是利大于弊的，不过需要合理地使用压缩，通过选择对一定范围大小的组件进行压缩和选择要压缩组件的类型，能使得收益最大化。

- 1 优点。压缩组件可以减少HTTP响应时间，提升传输效率。同时因为传输内容的减少，进而减少了带宽成本，通常能节省40%以上的带宽成本。
- 2 缺点。服务器要通过花费额外的CPU周期来完成压缩，客户端要对压缩文件进行解压缩。对于图片而言，却不应该对图片进行压缩，因为图片自身需要专业图片优化工具处理，如果再进行Gzip压缩，有可能得到的结果是和图片本身大小相差不大或更大，这样就浪费了服务器的CPU资源来做无用功了。不是Web服务器开启压缩一定会提高性能，因为每次压缩，都需要对文件进行压缩算法，将会消耗一定CPU和I/O的。因此，当文件较小时没有必要开启压缩功能。

网页加载速度加快的好处不言而喻，除了节省流量，改善用户的浏览体验外，另一个潜在的好处是Gzip与搜索引擎的抓取工具有着更好的关系。例如 Google就可以通过直接读取Gzip文件比普通手工抓取更快地检索网页。在Google网站管理员工具（Google Webmaster Tools）中你可以看到，sitemap.xml.gz是直接作为Sitemap被提交的。而这些好处并不仅仅限于静态内容，PHP等动态页面和其他动态生成的内容均可以通过使用Web服务器压缩模块压缩。

6.3.2 缓存优化

HTTP缓存机制定义在HTTP协议标准中，被现代浏览器广泛支持，同时也是用于提升Web性能

的重要途径之一。客户端经常会访问同一个资源，比如通过浏览器访问网页，或用APP访问同一个API，如果该资源和之前访问过的内容没有任何改变，可以直接从客户端的缓存中将内容返回给用户，而无需在服务端再生成一次内容，同时还减少一次内容传输时间。所以通过对浏览器请求的静态或动态资源的本地缓存，可以提升页面响应速度及降低带宽成本的作用。

HTTP缓存

浏览器在首次向服务器端发起请求时，会下载大量的网页资源，而这些资源并不是每时每刻都在更新的，对于那些不经常改变的静态资源，比如CSS、图片、动画等，应尽可能地利用缓存。因为这些资源通常很大而且几乎每个页面可能都会用到，缓存会大大提升用户体验。所以我们需要把这些资源缓存下来，以减少重复的HTTP请求，缓存过程如图6-32所示。通常会通过分析用户访问日志，优先对访问量大且流量高的网页资源进行缓存优化。HTTP缓存的主要有以下几类。

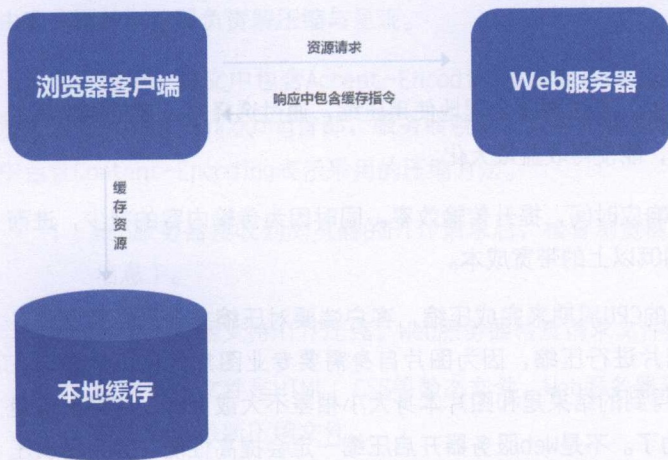


图6-32 缓存示意图

- 1 Cache-Control, HTTP1.1引入了Cache-control头来克服Expires头在服务器与客户端时钟不同步上的问题，并且如果使用Expires头，当到期日来到时，还需要在服务器配置中提供一个新的日期。Cache-control头使用max-age指令来指定组件被缓存多久，这是一个以秒为单位的相对时间段。当对于不支持HTTP1.1的浏览器，你可能需要同时提供Expires和Cache-control，当两者同时出现时，HTTP规范指出，max-age将重写Expires头。
- 2 Expires 是相对于max-age的另一种指示过期时间的方式。max-age表示多少时间后过期，而Expires表示在某个日期时间点后过期。换种通俗的说法，max-age相当于说“保质期六个月”，而Expires是说“在此日期之前”饮用。如果max-age和Expires同时存在，应该以max-age为准。但是通常可以把两个都设置成一个时间点。Web服务器通过使用Expires头

来告诉Web客户端，它可以使用一个组件的当前副本，直到指定的时间为止。Expires头包含了一个未来某一时刻的过期时间。通过这种方式，可以使浏览器在过期之前，都不再发送对这个组件的HTTP请求。但是这种方式存在一个客户端和服务器的时钟同步的问题。另外，Expire过期时间不能设太短，这样会导致由于客户端时间不一致带来的问题。

- 3 Last-Modified 和 If-Modified-Since Last-Modified出现在响应中，告诉浏览器当前资源的最后修改时间。而If-Modified-Since出现在下次请求中，询问服务器当前缓存的资源是否已经过期。如果资源没有过期，则服务器应该返回304HTTP状态码，不需要返回资源的内容。如果已过期，则服务器应该返回资源的内容到浏览器，并且返回200HTTP状态码。Last-Modified和If-Modified-Since配套使用，可以在保证不会误用缓存里的过期资源的前提下，减少服务器向浏览器发送的数据量，以及由于服务器在缓存未过期的情况下只需要返回304 HTTP状态码，而不需要返回整个资源的内容，也给了服务器优化的机会。
- 4 ETag和If-None-Match, Last-Modified 和 If-Modified-Since是用日期时间判断一个缓存的资源是否有效。ETag和If-None-Match则是用内容摘要作为判定的依据。内容摘要是指为一个资源的内容产生一串比较短的数字，当内容变化时，产生的数字串也会改变。内容摘要的算法有很多种，较常见的是SHA-1哈希算法、CRC32等。ETag包含在服务器的响应中，为内容摘要。在下次请求中，If-None-Match的值为上次的ETag的值。服务器根据If-None-Match的值（即内容摘要）判断缓存的资源是否有效。HTTP1.1引入了E-tag。E-tag是唯一标识一个组件的一个特定版本的字符串，需要用引号括起来。如果浏览器需要匹配验证一个组件，那么它会使用If-None-Match头将E-tag传回原始服务器，如果匹配则返回304状态码。E-tag与更具最新修改日期的验证更为灵活，但也带来了更多的问题。通常组件的E-tag使用组件的某些属性来构造，这些属性对于特定的服务器来说是唯一的，当浏览器从一台服务器上获取了原始组件，又向另外一台服务器来发起验证时，E-tag就不会匹配。同样由于E-tag的不匹配，还会造成的是代理缓存效率的低下。

DNS缓存

通常在一个浏览器向一个服务器发起请求之前，需要首先查找一个给定的主机名的IP地址需要花费20~120ms，在每次DNS查找完成之前，浏览器不能从服务器主机下载到任何东西。一般情况下，操作系统和浏览器本身都会对DNS的查询结果进行缓存，但各个不同浏览器之间的实现方案不同，特别是在DNS查询结果的过期时间TTL这一点上。通过DNS缓存，使得再向已缓存的主机名发出请求时，不再需要等待DNS的返回结果，并对页面中出现的新域名进行预解析，可以通过两个方面：

- 1 尽量使用keep-alive，这可以通过重用TCP/IP连接，减少开销，减少响应时间。

- 2 减少独立的主机域名数，但是减少独立主机域名数，会潜在地减少页面中并行下载的数量，这样可能拖慢了响应时间。因此一般的推荐建议是在两者之间找到一个平衡，一般希望的域名数是至少2个，但不要超过4个，这是在减少DNS查找和允许高度并行下载之间的权衡。

应用缓存

缓存数据库的查询结果，减少数据库的压力。通常来说Web应用性能瓶颈都出现在DB IO上，做好数据查询缓存，减少数据库的查询次数，可以极大提高整体响应时间。通常缓存承担了80%以上的数据访问请求。通常通过分析Rails、MySQL等日志，重点分析慢查询及针对慢查询进行重点缓存优化。使用连接池来减少连接次数及使用缓存来避免重复的数据库查询，同时使用索引来提升查询速度。

6.3.3 分离优化

服务分离及模块化是所有大型网站必须经历的过程，有一定用户规模的产品都需要针对业务类型进行模块化和剥离，例如门户资讯类网站都会将静态进行了最小颗粒的区分成模块，例如视频、图片、页面、下载模块等，每个模块都有对应的IDC分布策略、硬件选型、架构设计、容灾调度等，有新的产品需要上线，只要将内容打散分别接入对应的模块即可，速度和性能相比单独部署要优越很多。腾讯、百度的多个社区产品、商业化产品的性能优化也是经过服务拆分，进而才具备将速度做到极致的前提条件。

拆分的基本原则

微服务（Microservices）是一种松耦合的，面向服务的架构。将一个应用拆分成多个独立的服务，每个服务都具有业务属性，并且相对独立地被开发、测试、部署。从另一个角度看，微服务的本质，是用一些功能比较明确、模块比较精练的服务拆分，分别采用相对独立的服务对各方面进行管理，彼此之间使用统一的接口来进行交流，架构变得复杂，优势也很明显。再有，耦合性与内聚性也是两个重要定性标准，将应用系统划分模块时，尽量做到高内聚低耦合，提高模块的独立性，为设计高质量、高性能的系统结构奠定基础。

- 1 耦合也称块间联系，指软件系统结构中各模块间相互联系紧密程度的一种度量。模块之间联系越紧密，其耦合性就越强，模块的独立性则越差。模块间耦合高低取决于模块间接口的复杂性、调用的方式及传递的信息。
- 2 内聚又称块内联系，指模块的功能强度的度量，即一个模块内部各个元素彼此结合的紧密程度的度量。若一个模块内各元素联系的越紧密，则它的内聚性就越高。

根据业务划分

具备一定用户规模的互联网产品，都可以根据业务进行拆分，根据业务规模，还可以进行再拆分。然后针对性的定制硬件、Web Server、架构、运维自动化等工作。静态应用需要具备高负载、核心IDC+多家CDN分流、IDC容灾、区域容灾等能力，将热点数据留在离用户最近的接入层服务器的内存中，冷数据做自动淘汰，以便实现最小成本，最快腾挪。动态应用要求高性能、可伸缩、高内聚、低耦合等。细节如图6-33所示。

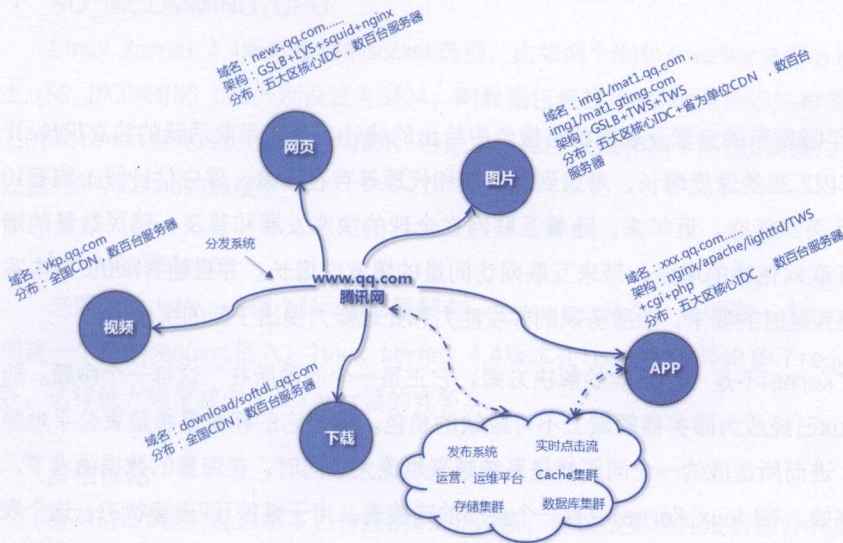


图6-33 腾讯网架构和运营策略

根据应用划分

软件架构是一个包含各种组件的系统组织，这些组件包括Web服务器，应用服务器，数据库，存储，通讯层等。个人的体会，无论产品形态，无论规模，系统架构既不是越大越好，也不是越多越好，而是尽可能从硬件层、系统层、数据库层、应用层、网络层等分层考虑，每一层功能尽可能单纯，而且高性能。如果一个模块太复杂，就或多或少会超过自身的能力控制范围，系统低耦合才容易扩展，足够简单才容易控制，如图6-34所示，通常将系统分成若干层，每一层分成若干模块。

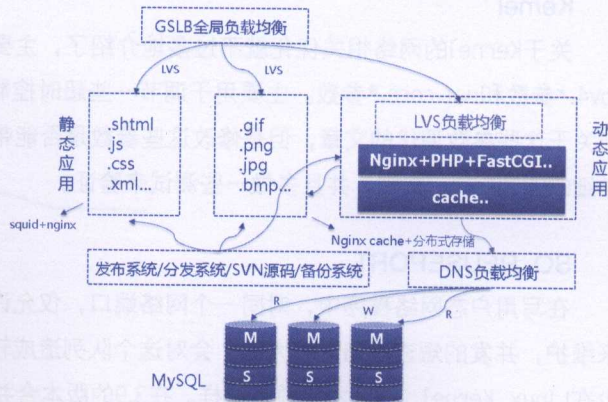


图6-34 应用模块化示意图

系统每一层，每一模块又互相关联拼接成一个整体，但每一层和模块相互独立透明，用户通过GSLB调度进入到负载均衡，从而分流到不同的应用层，应用层通过代理或内网负载均衡访问数据库层，每一层是独立且上下流关系，每一层都有容灾及故障迁移能力。每一个模块集群因海量用户及体验、容灾备份需要，必须做多地域、多运营商分布，但每一层及每一层模块之间的交互都需要尽可能在本地完成，跨区域交互需要专线、Qos保障，分布式存储和接入模块及用户访问、上传、加工都在本地完成，再汇聚到中央存储和备份。

6.3.4 内核优化

根据Akamai 2015年Q3发布的全球互联网状态报告中给出的统计，全球目前活跃的独立IPV4 IP数在8.08亿左右且每年以2.3%的速度增长，考虑到防火墙和代理等存在因素，保守估计线上约有10亿的主机保持着互联互通的状态。近年来，随着互联网在全球的快速发展和普及，网民数量的增加，生活中各方面对互联网依赖的增强，带来互联网访问量的爆发性增长。并且随着Web页面内容元素越来越丰富，对交互延时的要求，给服务端的并发能力和处理能力提出了新的挑战。

早有人已经提出“Kernel不是一个万能的解决方案，它正是一个问题所在”这样一个命题。随着互联网的发展，Linux已经成为服务器领域上不可或缺的角色。但是它的初衷是希望更公平地服务众多的用户和场景，进而所造成的一个问题就是系统越来越庞大。同时，在海量的数据请求下，瓶颈也越来越明显。例如，在Linux Kernel中有一个全局的连接表，用于维护TCP连接状态，这个表在维护大量的TCP连接时，会造成相当严重的资源竞争。在很多To C产品接入服务器上也发现大部分CPU资源消耗在Kernel里，并且在多核平台下，Kernel在网络协议栈处理过程中存在着大量同步开销。

Kernel

关于Kernel的网络相关优化就不过多地介绍了，主要的内核网络参数的调整在以下两处：net.ipv4.*参数和net.core.*参数。主要用于调节一些超时控制及缓存等，通过搜索引擎我们能很容易找到关于这些参数调优的文章，但是修改这些参数是否能带来性能的提升，或者会有什么弊端，建议详细的阅读kernel文档，并且多做一些测试来验证。

SO_REUSEPORT

在写用户态网络程序中，对同一个网络端口，仅允许一个监听实例，接收的数据包由一个队列来维护，并发的短连接请求较大时，会对这个队列造成较大的竞争压力，成为一个很大瓶颈点，至少在Linux Kernel 3.9版本之前是这样。在3.9的版本合并了一个很关键的特性SO_REUSEPORT，支持多个进程或线程监听相同的端口，每个实例分配一个独立的队列，一定程度上缓解这个问题。

用更容易理解的角度来描述,就是支持了在用户态上对一个网络端口,可以有多个进程或线程去监听它。正是因为有这样一个特性,可以根据CPU的核心数量来进行端口监听实例的选择,进一步优化网络连接处理的性能。每个listener只有一个accept队列, reuseport通过多个不同的socket文件描述符listen在同一个地址端口,增加可以竞争的锁的数量,提升并发度,避免accept锁和惊群,并且内核提供负载均衡能力。

SO_INCOMMMING_CPU

Linux Kernel 4.4版本提供的socket选项,比如两个Nginx worker进程分别绑定到cpu-3, cpu-4上, SO_INCOMMMING_CPU分别设置为3和4,则数据包查找listener全局表的时候,只会找到在对应CPU上的listener(避免reuseport的均衡),通过应用程序设置和网卡队列的映射,结合reuseport可以高效提升CPU Cache的热度。

lockless listener

老版本的内核,在每个listener里都有一个request队列,每收到一个syn包后都要锁listener并创建一个新的request插入。linux kernel 4.4版本在listener之外维护了request,使用TCP的ehash表,这样就大幅度减少了listener锁的竞争。

多核优化

面对的CPU核数越来越多,传统的Linux内核,协议栈实现并没有很好的支持多核,大量的数据共享使我们不得不使用大量的锁来控制并发,我们期望系统的性能同CPU核数成线性相关,需要的是一个像高速公路的架构而不是一个靠红绿灯控制的十字路口的架构。在多核上进行测试,HTTP CPS(Connection Per Second)吞吐量并没有随着CPU核数增加呈现线性增长,如图6-35所示,因为内核实现的限制,多个核心会竞争一些全局性的锁,比如listen socket锁。

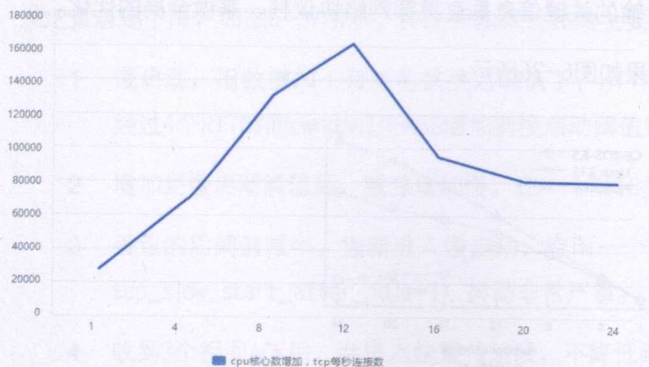


图6-35 CPU数量增加与连接数变化图

通过网卡多队列的支持,就可以有效利用多核心。网卡收到数据包后,把相同的TCP数据流发送到同一个CPU上,同时可以设置应用程序的CPU亲和性绑定到CPU。这样同一个TCP流都在同一个CPU上处理,大幅度提升CPU Cache的命中。如果服务端开启了irqbalance服务,系统会根据CPU的负载来动态的分发网卡中断,因此如果自己指定整个TCP处理路径,需要停止该服务。因此多核优化的原则主要如下:

- 1 相同的数据流在相同的CPU上处理,避免cache bouncing。
- 2 尽量使用percpu本地变量,避免或减少全局锁的粒度。
- 3 CPU负载均衡,在保持CPU本地化处理的基础上,避免单核压力过高。

Fastsocket

Fastsocket是新浪开源的,并在生产环境部署的一个内核优化版本。主要在于提高了Kernel网络协议栈的效率,所以网络I/O密集的应用可以收到很好的性能提升效果。Fastsocket对网络协议栈内部优化,可以分为两个维度。

- 1 多核扩展性的优化,也就是让Kernel网络协议栈在多核平台发挥多核的并行处理优势。这个维度又可以分为两个方向:一是,将网络协议栈处理的关键数据结构做CPU核心间的隔离,使得每个核心有完全本地的访问数据,从而消除了执行路径上核心间的同步开销。二是,使得任意的某个TCP连接的全部处理,都在一个核心上完成,这样可以最大化的提高CPU Cache利用率。
- 2 单核性能的提升,也就是不考虑多核同步的情况下,如何提升网络协议栈在单个核心上的绝对性能。这个维度也可以分为两个方向:一是,将Kernel中的通用服务为网络I/O做专用定制,来提升网络协议栈的性能。二是,做网络协议栈的跨层优化,改变传统协议栈TCP/IP协议栈的严格分层处理,将传输的关键信息垂直贯穿网络协议栈,来做全局的优化。

使用了 Fastsocket 后的性能测试结果如图6-36所示。

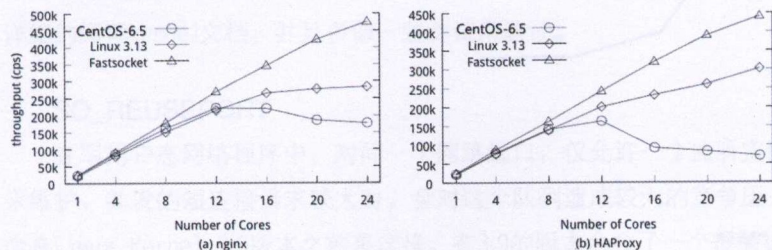


图6-36 Fastsocket性能测试数据

Fastsocket比较适用于一下场景：

- 1 系统至少不少于8个CPU。
- 2 系统的很大一部分开销用于处理网络软中断以及socket相关系统调用。
- 3 TCP短连接很多。
- 4 应用使用了epool处理网络IO。
- 5 应用使用了多进程接收连接。

6.3.5 传输优化

根据发布的第36次中国互联网络发展状况统计报告数据显示，其中各类点对点技术、高清视频、娱乐、直播应用呈现爆发式增长；服务器、带宽等成本占据此类企业运营开销的大头。横向上可以增加机器来提高服务的可用性和解决并发量的增长，虽然硬件越来越便宜，但是一味增加机器会造成大量的维护成本和非高峰时候资源的闲置。结合对现有机器的优化和合理的资源配置，提高单台机器的新建连接并发能力，减少访问延时，提高网络带宽利用率已经迫在眉睫，其中拥塞控制就重中之重。

传统拥塞算法弊端

绝大多数的TCP实现都是基于TCP New Reno 及其变种（例如 TCP SACK, RFC 3517）作为其拥塞避免算法。New Reno的拥塞避免算法是基于丢包统计的算法。基于丢包的算法，将丢包作为网络发生拥塞的标志。但是，随着Internet的发展，这个假设在现在的网络环境下，很多时候并不符合实际情况，特别是在移动网络中，丢包的原因通常不是链路拥塞，而是由于信道衰减、无线的噪声等原因导致。New Reno算法一旦发现网络上发生丢包，就会将拥塞窗口（CWND）迅速缩小，导致数据发送量急剧下降，如图6-37所示。传统拥塞算法弊端主要如下：

- 1 慢启动，指数增加（有可能被推迟确认），HTTP/2.0通过减少连接数来避免慢启动，下图经过4个RTT时间cwnd从1个MSS增加到慢启动阈值16。
- 2 增加到慢启动阈值后，线性增加慢。在一个RTT时间内收到所有确认只能增加1个MSS。
- 3 丢包的后阈值减半，重新进入慢启动，空闲一个重传超时RT0后也会重新慢启动(net.ipv4.tcp_slow_start_after_idle=1)，抖动非常严重。
- 4 收到3个相同ACK后，会进入快重传阶段，不降低阈值，线性增加。
- 5 经常性的慢启动和降阈值不但造成的抖动，还会造成带宽利用率大幅度降低。

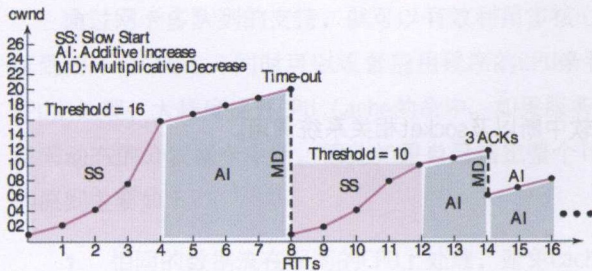


图6-37 拥塞窗口随丢包而缩小

商业公司的拥塞算法

1 FAST TCP, FAST TCP通过增加延时作为拥塞反馈信号, 建立更好的数学模型, 更快达到网络最大利用率(减少慢启动和线性增加所做的逐步探测网络带宽的过程)。根据丢包和延时动态计算一个网络的均衡点(在这个点实现了网络利用率的最大化, 并且能尽量保证各方的公平性)。离均衡点越远, 拥塞窗口调整越快, 否则越慢, 以下测试数据可以看到FAST TCP的带宽利用率最高, 抖动最少, 如图6-38所示。FAST TCP后被Akamai (全球最大CDN服务商) 收购。

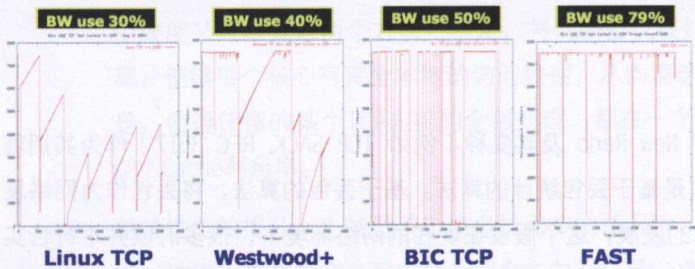


图6-38 FAST TCP提速效果

2 Zeta TCP, Zeta-TCP是华夏创新推出的基于传输历史学习的算法, 将延时和丢包的因素组合作为网络拥塞的衡量标准。Zeta-TCP不断测量丢包和延时的变化幅度并根据该TCP连接历史数据分析判断当前所经历的丢包, 延时及其变化是否由拥塞引起, 并据此来调整拥塞窗口, 从而让TCP的发送方流量最贴近TCP连接整条路径的可用带宽, 从而减少拥塞, 提高整条路径带宽利用率, 最终达到提高TCP传输稳定性和吞吐率的目的。

一些拥塞算法优化实践

1 引入更精确的丢包判断及预测算法。传统TCP经常错误判断丢包, 将未丢的数据包判断为丢失将导致错误的重传, 从而造成带宽的浪费。反之, 如果不能及时判断确实已丢失的数据包, 将导致不必要的等待, 从而导致带宽空置。两种误判都会降低连接的吞吐率和带宽的利用率。单边加速引擎能够精确及时地判断丢包, 从而保证了最佳的带宽利用率。

- 改进TCP的重传算法，优化流量巨量脉冲的问题，提高丢包重传时网络的稳定性。
- 随时精确侦测连接路径带宽，并相应调整发送数据量。TCP协议通过滑动窗口机制对带宽进行自适应。传统TCP的滑动窗口实现经常误判路径带宽容量。高估带宽容量将导致过量传输从而引发拥塞并导致大量丢包，低估带宽容量则导致闲置带宽容量，两个极端的出现都将导致带宽利用率的下降。传统TCP往往在两个极端之间震荡，很难有效利用带宽。我们的单边加速技术在主动精确侦测路径带宽的基础上，随时调整发送数据量，从而在防止引入拥塞的同时最大限度地利用路径带宽。
- 基于学习的拥塞判断及处理技术，通过对路径上传输历史的学习动态判断拥塞并调整传输速率。该拥塞判断及处理技术能够进行准确的延时和带宽估计、能够在拥塞避免阶段和丢包重传阶段快速逼近最优窗口值、能够准确地预估大延时网络的吞吐量和路由器队列长度。

基于以上拥塞算法优化后，分别测试全国下载80KB、150KB、2MB的文件，得到测试数据如下。

- 80KB文件测试，“总下载时间”优化前为0.48s，优化后为0.36s，较优化前提速25%，如图6-39所示。

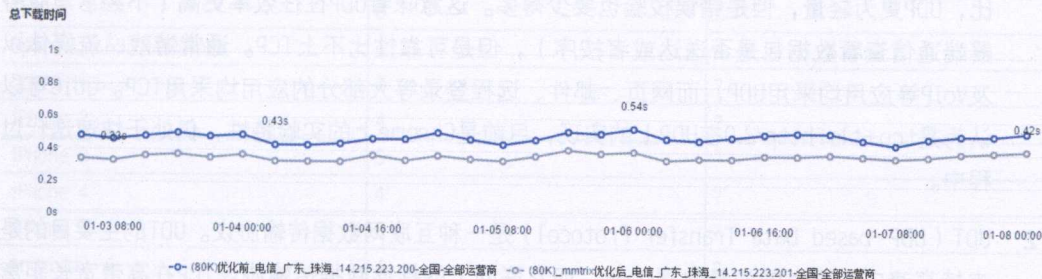


图6-39 80KB文件提速效果

- 150KB文件测试，“总下载时间”优化前为0.62s，优化后为0.45s，较优化前提速27.4%，如图6-40所示。

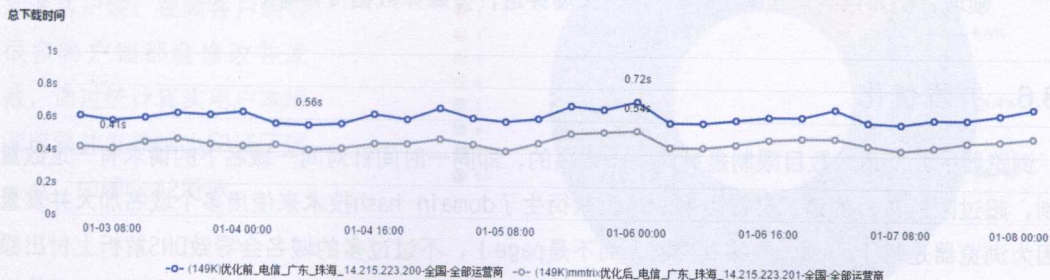


图6-40 150KB文件提速效果

- 2MB文件测试，“总下载时间”优化前为3.36s，优化后为2.57s，较优化前提速23.5%，如

图6-41所示。

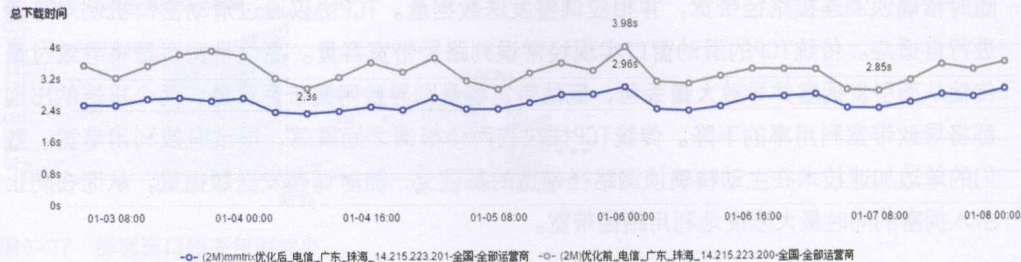


图6-41 2MB文件提速效果

另外，由于内核TCP实现的一些问题存在，所以目前大多数双边加速都是在UDP的基础上，增加应用层的TCP的可靠性机制（确认，重传，按序等）。因为客户端和服务端都需要调用该协议SDK，如果要对现有的软件接入该SDK，基本上需要对软件进行重构和重写，因此开发成本较大。

- 1 QUIC (Quick UDP Internet Connection) 是谷歌制定的一种基于UDP的低时延的互联网传输层协议。TCP/IP协议是互联网的基础。其中传输层协议包括TCP和UDP协议。与TCP协议相比，UDP更为轻量，但是错误校验也要少得多。这意味着UDP往往效率更高（不经常与服务端通信查看数据包是否送达或者按序），但是可靠性比不上TCP。通常游戏、流媒体以及VoIP等应用均采用UDP，而网页、邮件、远程登录等大部分的应用均采用TCP。QUIC可以认为是tcp+tls+http/2.0在UDP上的实现，目前是Chrome上的实验特性，仍处于快速迭代过程中。
- 2 UDT (UDP-based Data Transfer Protocol) 是一种互联网数据传输协议。UDT的主要目的是支持高速广域网上的海量数据传输，而互联网上的标准数据传输协议TCP在高带宽长距离网络上性能很差。顾名思义，UDT建于UDP之上，并引入新的拥塞控制和数据可靠性控制机制。UDT是面向连接的双向的应用层协议，它同时支持可靠的数据流传输和部分可靠的数据报传输。由于UDT完全在UDP上实现，它也可以应用在除了高速数据传输之外的其他应用领域，例如点到点技术（P2P），防火墙穿透，多媒体数据传输等。

6.3.6 并发优化

浏览器的并发请求数目限制是针对同一域名的，即同一时间针对同一域名下的请求有一定数量限制，超过限制数目的请求会被阻塞。这也就衍生了domain hash技术来使用多个域名加大并发量（因为浏览器是基于domain的并发控制，而不是page），不过过多的域名会导致DNS解析上付出额外的代价，所以一般根据实际情况控制在2-4之间。这里常见的一个问题是没有机制确保URL的绝对分布，从而导致资源分布到域名上是不均匀的。

浏览器并发限制

不同的浏览器对单个域名的最大并发连接数有一定的限制，HTTP/1.0和HTTP/1.1也不相同。当浏览器与服务器所建立的并发连接的数量达到浏览器所设定的最大限制时，浏览器会将新增的连接请求进行阻塞，并将其加入到浏览器内置的请求等待队列中，待有请求从服务器返回时才从等待队列中选取队首请求发送。一些主流浏览器对 HTTP 1.1 和 HTTP 1.0 的最大并发连接数目，可以参考如表6-6所示。

表6-6 浏览器并发统计表

Browser	HTTP/1.1	HTTP/1.0
IE 11	6	6
IE 10	6	6
IE 9	10	10
IE 8	6	6
IE 6,7	2	4
Firefox	6	6
Safari 3,4	4	4
Chrome 4+	6	6
Opera 9.63,10.00alpha	4	4
Opera 10.51+	8	?
iPhone 2	4	?
iPhone 3	6	?
iPhone 4	4	?
iPhone 5	6	?
Android2-4	4	?

往往真实用户本地浏览上的并发数与浏览器没有绝对的联系，像下载客户端、加速客户端、视频客户端等很多客户端都会修改并发数，通过统计真实用户本地浏览器并发数据也印证了这一点，如图6-42所示。

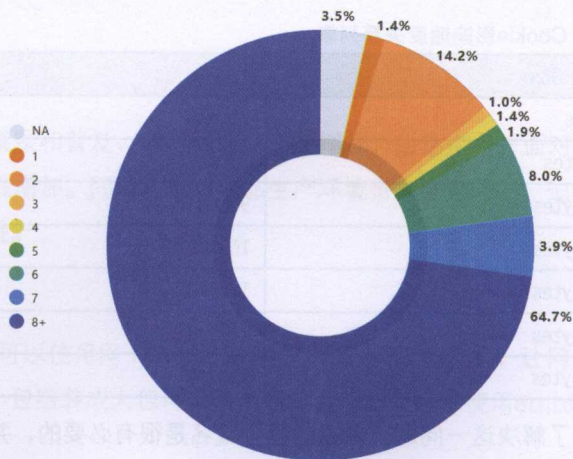


图6-42 真实用户并发统计

优秀企业的经验

表6-7 各企业域名拆分统计表

企业	产品	静态域名数量	域名
阿里	淘宝	2	img/gw.alicdn.com
阿里	天猫	2	img/gdp.alicdn.com
腾讯	门户	2	img1/mat1.gting.com
腾讯	空间	4	a1~4.qpic.cn
百度	搜索	4	ss0~ss3.bdstatic.com
京东	官网	5	img10~14.360buyimg.com
小米	商城	3	i1~i3.mifile.cn
新浪	微博	4	ww1~ww4.sinaimg.cn
搜狐	门户	4	i0~i3.itc.cn
网易	门户	5	img1~5.cache.netease.com

6.3.7 隔离优化

Cookie作为一种在Web开发中常使用进行本地存储的方法，已被广泛使用。但是使用Cookie方法在浏览器中进行本地存储具有以下一些不可忽视的问题，每次客户端再与服务器进行通信交互时，无论是否需要Cookie，Cookie都会随着客户端的HTTP请求一起发给服务器端，这样势必影响请求响应速度及浪费带宽。例如当网站Cookie信息有1KB、网页共150个网页元素时，用户在请求过程中需要发送150KB的Cookie信息。尽管这个过程可以和页面下载不同资源的时间并发，但毕竟对速度造成了影响。而且这些信息在js/css/images/flash等静态资源上，几乎是没有任何必要的，这就是所说的Cookie污染，Cookie对速度的影响如表6-8所示。

表6-8 Cookie影响速度关系列表

Cookie Size	Time	Delta
0 bytes	78 ms	0 ms
500 bytes	79 ms	+1 ms
1000 bytes	94 ms	+16 ms
1500 bytes	109 ms	+31 ms
2000 bytes	125 ms	+47 ms
2500 bytes	141 ms	+63 ms
3000 bytes	156 ms	+78 ms

为了解决这一问题，单独的多个域名是很有必要的，并且不受Cookie大小和个数的限制，而且在客户端与服务端的每次交互请求的时候，不再会有Cookie信息跟随着HTTP请求，降低了请求的大

小，不但提升了速度，还减少了带宽成本。以前腾讯网所有的网页元素都使用qq.com域名，每个请求都带了Cookie，而且较大，大小有500B~2KB。采用非主域名的图片和素材服务器，可以有效地隔离Cookie，加快每个请求的响应速度，如图6-43所示。

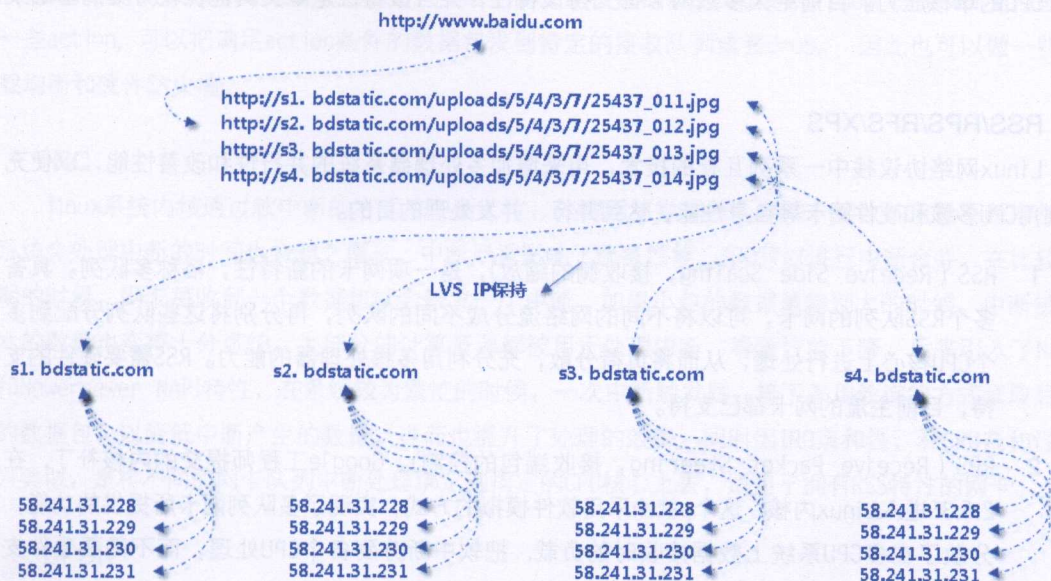


图6-43 Cookie隔离收益图

另外，大部分浏览器对单个域名下的Cookie的个数和Cookie的总的大小有着限制，一般为4k大小，这样当你需要存储的内容大于这个大小，就会影响存储信息。所以最小化Cookie大小，使用最短的Cookie名和Cookie值是非常有必要的。

6.3.8 网卡优化

近年来，随着互联网在全球的快速发展和普及，网民数量的不断增加，生活中各方面对互联网依赖的增强，带来互联网访问量的爆炸性增加。同时，也给产品生产环境带来巨大冲击，迫使我们单机下高并发的网络承载能力进行改进。

CPU offload

对于大量小包为主CPU密集型场景，可以使用网卡的offload特性，减小CPU的负载，让网卡计算校验和，利用网卡来执行TCP分段，将小包组装成大包再交给内核协议栈等，可以使用ethtool工具来开启相应特性。

利用网卡MSI-X多队列特性

MSI方式的中断对CPU多核的利用不佳，网卡中断全都落在某个CPU核心上，MSI-X方式可以为每个队列申请一个中断号，然后设置中断亲和性，把该队列的中断映射到某个特定的CPU核心上，分担了CPU的单核压力。目前绝大多数网卡都支持该特性，并且该特性是本文其他优化方案的基础条件。

RSS/RPS/RFS/XPS

Linux网络协议栈中一系列互补的技术，用来增加多处理器系统的并行性和改善性能，以便充分利用CPU多核和硬件网卡等自身性能，达到并行、并发处理的目的。

- 1 RSS (Receive Side Scaling, 接收侧的缩放), 是一项网卡的新特性, 俗称多队列。具备多个RSS队列的网卡, 可以将不同的网络流分成不同的队列, 再分别将这些队列分配到多个CPU核心上进行处理, 从而将负荷分散, 充分利用多核处理器的能力。RSS需要硬件的支持, 目前主流的网卡都已支持。
- 2 RPS (Receive Packet Steering, 接收端包的控制)。Google工程师提交的内核补丁, 在2.6.35进入Linux内核。这个patch采用软件模拟的方式, 实现了多队列网卡所提供的功能, 分散了在多CPU系统上数据接收时的负载, 把软中断分到各个CPU处理, 而不需要硬件支持, 大大提高了网络性能。
- 3 RFS (Receive Flow Steering, 接收端流的控制)。Google工程师提交的内核补丁, 它是用来配合RPS补丁使用的, 是RPS补丁的扩展补丁, 它把接收的数据包送达应用所在的CPU上, 提高cache的命中率。通常与RPS一起设置, 来达到最好的优化效果, 主要是针对单队列网卡多CPU环境(多队列多重中断的网卡也可以使用该补丁的功能, 但多队列多重中断网卡有更好的选择:SMP IRQ affinity)。当应用程序调用recvmsg的时候, 会记录所在的CPU到hash表中, hash表的key是网卡RSS计算的hash值。因此下一次数据包来的时候就可以直接发到应用程序所在CPU进行处理。
- 4 XPS (Transmit Packet Steering, 发送端包的控制)。Google工程师提交的内核补丁, 建立CPU内核和Tx发送队列映射对应关系, 掌控出站数据包。系统有N个CPU核心, 脚本会设置XPS至少存在N个Tx队列在网卡接口上, 这样就可以建立CPU内核和Tx队列1对1的映射关系。网卡发送队列发送数据包完成后, 可以中断到特定的CPU上, 因此可以让更少的CPU竞争该发送队列, 减少中断造成的Cache miss减少。

Intel Ethernet Flow Director

RSS解决了CPU的单核心负载高的问题, 把不同的数据流分发到不同的CPU上, 但是有可能

应用程序并不在该CPU上。因此Flow Director就是因此而来，它有两种模式：EP（Externally Programed）模式和ATR（automated Application Targeting Routing）模式。如果了解整个网络主要的数据包路径的时候可以配置EP模式，否则应使用默认的ATR模式。ATR模式会采样发送流量，然后根据TCP头的来源与目标信息把数据包交给之前发送该数据流的CPU核心。Flow Director支持配置一些action，可以把满足action条件的数据包发到特定的接收队列或者drop。因此也可以做一些负载均衡和硬件防火墙。

IRQ

linux系统内核通过软中断的方式来中断工作，处理网络数据包。当网卡接收的流量不断增加，系统会处理中断的时间也会随之增长，中断严重影响了服务性能。IRQ可以进行中断合并，在比较早期的时候，网卡每收到一个数据包就会触发一个中断，如果小包的数据量特别大的时候，中断被触发的数量也变得十分可怕。大部分的计算资源都被用于处理中断，导致性能下降。后来引入了NAPI和Newernewer NAPI特性，在系统较为繁忙的时候，一次中断触发后，接下来用轮询的方式读取后续的数据包，以降低中断产生的数量，进而也提升了处理的效率。同时因IRQ亲和性，和CPU亲和性较为类似，是将不同的网卡队列中断处理绑定到指定的CPU核心上去，适用于拥有RSS特性的网卡。

DPDK

内核本身不可避免的开销，例如系统调用，陷入内核态，上下文切换及内核态到用户态的数据拷贝、中断处理等，加上不容易调试，代码比较复杂等原因增加开发和维护难度。同时随着网络的爆发性增长和C10M概念的出现，基于数据面的实现（绕过内核协议栈）也逐渐成为了业界关注之一。DPDK是Intel提供的提升数据面报文快速处理速率的应用程序开发包，它主要利用以下几个方面的支持特点来优化报文处理过程，从而加快报文处理速率。

- 1 使用大页缓存支持来提高内存访问效率。
- 2 利用UIO支持，提供应用空间下驱动程序的支持，也就是说网卡驱动是运行在用户空间的，减下了报文在用户空间和应用空间的多次拷贝。
- 3 利用LINUX亲和性支持，把控制面线程及各个数据面线程绑定到不同的CPU核，节省了线程在各个CPU核来回调度。
- 4 提供内存池和无锁环形缓存管理，加快内存访问效率。

DPDK的应用架构如图6-44所示，可以看到dpdk应用为每个网卡队列启动了一个线程，每个线程绑定到一个CPU上，通过内核的uio驱动机制，网卡队列的数据包可以直接发送到用户态的缓存中，然后dpdk线程循环拉取该缓存，之后递交到用户态协议栈处理。

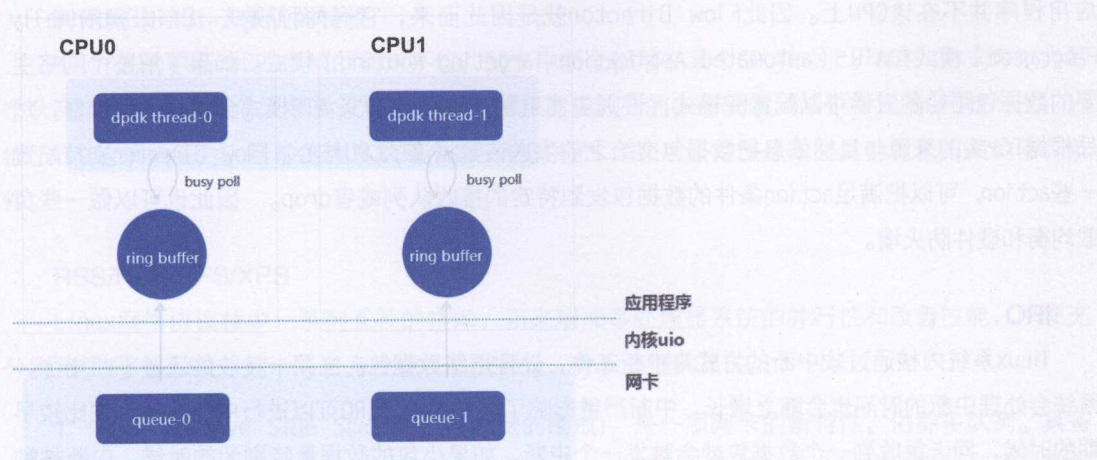


图6-44 DPDK示意图

6.3.9 硬件优化

硬件上的性能瓶颈，一般指的是CPU、内存、磁盘I/O、网络方面的问题，例如服务器硬件瓶颈、网络瓶颈（对局域网可以不考虑）、操作系统瓶颈（参数配置）、中间件瓶颈（参数配置、数据库、Web服务器等）、应用瓶颈（SQL语句、数据库设计、业务逻辑、算法等），加上生产环境服务器资源利用率是不均匀的，新、老业务及对应的服务器因历史原因同时存在或混合部署，导致系统资源分布和利用不均衡，性能差的服务器短板效应非常明显。

大多数情况下，虽然性能瓶颈的起因是程序性能差或者是内存不足或者是磁盘瓶颈等各种原因，但最终表现出的结果就是CPU耗尽，系统负载极高，响应迟缓，甚至暂时失去响应，因此我们观察服务器状况时，最先看的就是系统负载和CPU空闲度。当然，大多数的硬件性能问题主要和CPU、内存、磁盘相关，但是应用程序设计的缺陷和数据库查询的滥用也是最常见的性能问题。分享系统资源使用原则和经验如下：

- 1 对资源的使用状况做长期的监控和趋势分析。
- 2 程序的优化和系统结构的优化比硬件的性能优化更有效。
- 3 系统硬件配置以适合应用的特点为依据，以资源消耗均衡为目标，避免程序不受限制的使用系统资源。
- 4 从上到下架构梳理，从宏观到微观。不断审视和熟悉产品、架构、代码、算法、冗余等。（系统的优化程度，取决于对它的了解程度）

- 5 持续进行架构调优，服务分层及横向扩展，结构扁平化，每层服务对应的硬件配置尽可能一致。
- 6 能靠硬件先靠硬件。软硬件用好，不浪费而且事半功倍。新硬件，旧软件，肯定浪费。

CPU

- 1 深刻理解真实需求，简化实现方案。
- 2 一个好的架构，服务器的CPU总消耗总是平均的分布在各个CPU上，CPU的消耗在70%左右。
- 3 尽量不用锁，谨慎用锁。
- 4 慎用字符串操作，很多操作都需要CPU资源去做词法分析，数量多的话，也是不菲的开销。
- 5 减少系统调用，例如time，主要消耗在用户态和内核态之间的切换。
- 6 减少遍历操作，字符串协议解包很耗CPU，考虑单独一个线程。
- 7 单颗CPU的性能对依赖CPU的某些应用的影响很严重，比如数据库的查询处理。
- 8 使用及使用好更强的CPU。

内存

- 1 基于数据统计的内存优化，内存占用排名分析并持续优化。
- 2 谨防内存泄露(Valgrind)，避免内存频繁申请和释放(内存池)。
- 3 尽量减少内存拷贝(合理的程序结构更重要)，作用域和生命周期相同的数据放一起。
- 4 物理内存不够时会使用交换内存，使用交换内存会带来磁盘IO和CPU的开销增加。

磁盘I/O

- 1 从磁盘到SSD，积极进行硬件更新换代，发挥SSD特性（读取快、寿命短、价值贵），积极尝试Flashcache。
- 2 利用顺序写，减少寻道次数，单线程按Block对齐写入，读取不要小于一个Page。
- 3 小文件读写的性能瓶颈是磁盘的寻址（随机读写性能最差，小数据写入在内存合并），大文件读写的性能瓶颈是带宽，评估的标准是持续读写速度。
- 4 正确使用RAID磁盘阵列（RAID0, RAID1, RAID5, RAID0+1）。
- 5 Linux可以利用空闲内存作文件系统访问的Cache，充分利用内存的资源来缓解磁盘读写压力。

- 6 在Linux下可选的文件系统有ext2、ext3、xfs、ReiserFS，根据不同的应用，选择不同的文件系统。

网络

- 1 使用网卡硬件多队列，网卡中断和CPU绑定，使用SO_REUSEPORT，连接在单个线程处理。
- 2 秒级的网络监控，带宽消耗，包个数，包大小等。
- 3 使用epoll代替select，使用非阻塞的模式。
- 4 包个数优化，减少不必要的发包，减少重复发包，随机丢弃影响不大的包。
- 5 单个包协议的优化，根据默认内容不同的协议拆分，协议字段的节省，协议压缩。
- 6 基于客户端缓存的带宽优化方案。
- 7 根据情况进行预处理（压缩），增加CPU减少网络传输。
- 8 合理使用万兆网卡。

静态应用

- 1 网络带宽瓶颈。
- 2 小文件的随机读取频繁，内存充足时可以缓解磁盘随机读的压力。
- 3 系统内存不足时磁盘IO量会比较大（读、写、交换内存），因此增加CPU的开销。
- 4 存储系统带宽瓶颈（读）。

动态应用

- 1 频繁执行程序，如 Perl, PHP, Java 等，消耗CPU严重。
- 2 提供并发用户访问，因此系统进程数多，消耗内存多，当内存不足时，使用交换内存也会增加CPU的开销。
- 3 磁盘的写IO比较频繁（主要为随机写），比如生成Cache文件，更新Session文件等。
- 4 内存充足时读取的内容可以被Cache住，Cache的命中率和文件更新的频繁程度成反比，磁盘的读IO相对较小。

新技术的优势

硬件有一个有限的生命周期，有必要在硬件产商停止为过时设备提供部件和维修支持之前，进

行硬件更新。同时，硬件的新技术是势不可挡的，而且迭代速度越来越快。新服务器可以通过提供冗余组件，例如多电源或网络适配器来获得额外的弹性，改进的可用性与可靠性则可以减少宕机时间与恢复成本。不止如此，新的硬件技术更新带来的新功能和性能提升将为业务提供更强劲的性能和稳定性，从而为用户带来更好的用户体验。

硬件定制化趋势

在外界看来，Facebook是社交网络巨头，但事实上，该公司拥有自己的硬件团队，开发了高效能、低成本的服务器。当时该公司用户飞速增长，数据传输量猛涨，因此亟需质优价廉的高性价比服务器。现如今，Facebook已经具备了建立服务器制造、元部件以及供应链、自建数据中心等全方位的能力，而且也引起了其他互联网公司公司的共鸣，包括谷歌和亚马逊等巨头，已纷纷开发服务器，打造自家数据中心。国内BAT也是如此，都在致力于打造贴近自身业务需求且具备最佳性价比的服务器基础架构，主要有以下几个方向。

- 1 TCO (Total Cost of Ownership) 和定制化，TCO即总拥有成本从产品采购到后期使用、维护的总的成本。
 - 1) 定制化，引入ODM，如富士康，广达。
 - 2) 部件研究、定制，机柜、服务器、硬盘、内存、SSD、多核处理器。
 - 3) 硬件监控与修复，保障业务稳定性。
- 2 与OEM厂商合作，对外坚持多品牌，去差异化，对内屏蔽品牌信息，套餐化，基于单机定制，关注成本和功耗。OEM厂商如图6-45所示。



图6-45 硬件厂商

6.4 前端优化

互联网产品被产品经理 (PM) 设计出来，然后前端工程师 (FE) 和研发工程师 (RD) 进行生产，由运营和销售推上市场，再由运维工程师 (OP) 和网络工程师 (SYS) 保持稳定运行，这四个环节都会对速度造成影响，所以说，应用性能监测、分析、优化是一个系统工程，详细如图6-46所

示。涉及的角色、人、环节非常多，而前端工程师在产品研发过程中扮演了非常特殊的角色，前端基本决定了用户看到和使用产品的内容和结构，同时在应用性能优化过程中也是中坚力量。

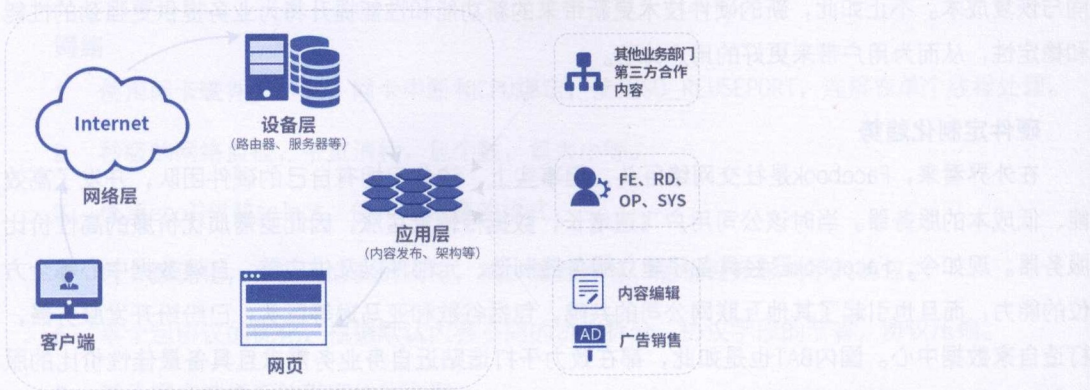


图6-46 应用性能优化多环节示意图

这里所说的前端泛指Web前端，也就是在Web应用中用户可以看得见碰得着的东西，包括Web页面的结构、Web的外观视觉表现以及Web层面的交互实现，同时也包含前端研发工程师、运营编辑等角色生产的所有网页内容。任何一个优化项目，前期网络层、系统层优化是立竿见影的，因为这两层做一个优化，整个产品都会受益，而且是一劳永逸。中后期主要优化工作在前端和内容，涉及页面和人多而且不断在迭代更新。将主要的前端优化实践汇总如表6-9所示。

表6-9 前端优化实践表

前端优化项	优化说明	备注
首屏优化	1. 用户访问网页的第一感知和关注点在网页第一屏，即首屏，首屏可以优化到0.5~2s，几乎立刻可见和可点击，大大提升了用户体验。 2. 最快速度显示页面的首屏内容，可以给用户明显的感知体验，首屏优化是前端优化之首	建议任何产品以首屏优化作为切入口，首屏优化目标是<2s
内容优化	在页面中图片、JS、Script、Flash等元素是页面和应用的主要构成，也是前端优化的重点，通常收益非常大	网页元素过多，是最常见的前端性能问题
请求优化	页面元素的数量、大小决定了页面加载时间和效率。页面和应用开发过程中，对请求的优化和规避是前端工程师的必修课	域名的数量与域名解析时间需要平衡，通常网页元素越多，域名的拆分收益越大
CSS优化	CSS样式表主要用于网页中样式的定义，是决定页面外在表现和性能的主要原因	通俗叫法为静态网页布局
JavaScript优化	JS串行加载等特殊特性，是响应页面性能的主要原因之一，而且比重较大，无论从大小、逻辑、数量上都需要严格控制	JS是被公认的性能“杀手”需要重点关注
图片优化	页面中，60%以上的请求和大小都是由图片构成，图片加载时间决定了页面加载的快与慢	图片优化伴随较大的成本优化

6.4.1 首屏优化

首屏的英文是above the fold, fold有折叠之意, above the fold是指在折叠之后能看到的, 为什么首屏的英文翻译会跟折叠有关系呢? 原因很简单, 因为这个概念最早用于出版领域, 可以简单地理解为“头版”因为报纸的运输和分发过程是折叠起来的, 所以报纸的折叠后暴露在读者面前的那一部分内容就显得尤其重要, 读者会根据头版的内容决定是否购买。因此处于头版的内容意味着一个, 编辑认为它们是最重要的, 头版的内容也决定了出版物的立场和定位。所以“above the fold”也用来表示所有优先显示或优先级最高的内容。

“above the fold”的概念延伸到互联网领域。用来指代Web网页中不用滚动屏幕看到的信息。与出版业的“头版”不同的是互联网的首屏区域是动态的, 由于互联网用户复杂的终端和屏幕分辨率环境, 导致他们看到的首屏内容有很大差距。很多产品就是因为对首屏的忽视导致很严重的体验问题。无论PC还是移动Web页面, 都受网络和终端及应用性能影响, 经常要关注首屏内容展示时间这个指标, 首屏如图6-47所示, 它衡量页面是否能在用户耐心消磨完之前展示出来, 很大程度影响着用户的使用满意度。



图6-47 首屏示意图

首屏时间的意义

世界著名的网页易用性报告显示，首屏以上的关注度为80.3%，首屏以下的关注度仅有19.7%。这两个数据足以表明，首屏对每一个需要转化率的网站都很重要，尤其是To C类网站，其他影响如下：

- 1 用户用视觉来感受网站的快慢。
- 2 尽早将重要的信息展现给用户。
- 3 残缺的信息展示用户同样会感觉网站慢。
- 4 尽早完整地展示首屏页。

首屏时间为1~2s是较好的，3s以内基本可以接受的，5s以上已经影响到用户的使用体验。根据strangeloop的调研，首屏时间对用户的影响非常重要，具体如表6-10所示。

表6-10 首屏时间对用户体验影响统计

首屏加载时间	页面浏览量(PV)	跳出率	转化率
3s	-22%	50%	-22%
5s	-35%	105%	-38%
10s	-46%	135%	-42%

首屏时间监测途径

首屏时间的统计比较复杂，因为涉及图片等多种元素及异步渲染等方式。现代浏览器边解析HTML边显示的实现，似乎让记录首屏时间变得困难，首屏监测途径及优劣对比如表6-11所示。

表6-11 首屏时间统计途径

途径	优势	劣势
时间戳判断	手动打时间戳判断首屏时间，在HTML文档中对应首屏内容的标签结束位置，使用内联的JavaScript代码记录当前时间戳	但是这只适用纯文本的内容，并且这种页面的首屏时间往往很小
根据图片判断	观察加载视图可发现，影响首屏的主要因素是图片的加载。通过寻找首屏区域内最后加载的图片作为首屏的结束时间，这样比较符合网页的实际体验。现代浏览器处理图片资源时是异步的，会先将图片长宽应用于页面排版，然后随着收到图片数据由上至下绘制显示的。并且浏览器对每个页面的TCP连接数限制，使得并不是所有图片都能立刻开始下载和显示	需要判断首屏有没有图片，如果没图片只能用domready时间替代，如果有图，还需要判断是不是在首屏内
读屏判断	根据自定义区域内，白屏之后屏幕像素变化到稳定这一过程判断第一屏填充完成，即首屏加载完成	干扰因素较多，例如轮播页、动画广告等导致首屏时间非常大
首屏时间=可交互时间	控制灵活，适合简单网页或纯文本	复杂页面完全不适用

首屏时间优化原则

首屏是吸引用户至关重要的通道，是被业界认可的页面最重要的性能指标，也是影响用户转化率权重最大的性能指标，通常通过以下几个方向进行首屏优化：

- 1 首屏最小化，首屏HTML尽量小，尽可能控制DOM节点数、请求数、外链数，让首屏的可视区域尽快显示。
- 2 首屏元素优化，优化落在首屏内的元素性能和结构，包括基础页、元素请求、图片、JS、是否调用第三方内容、层次结构等，这些都直接影响首屏时间。
- 3 页面静态化或全站网络加速，页面首屏包含了页面基础页时间（第一个请求），及屏内的元素总的DNS解析时间、建立连接时间、SSL握手时间、发出请求时间、重定向时间、内容下载时间等。
- 4 基础页优化，以静态页面的形式存放，用户相关数据依赖AJAX，比如登录信息。用户信息默认显示未登录状态，异步获取到用户数据后再更新。
- 5 首屏广告优化，重点减少广告JS的下载次数，减少状态上报次数，避免iframe。同时处理脚本放在页面底部，修改广告的载入顺序，避免影响内容显示。
- 6 首屏按需加载优化，隐藏tab页，用了异步加载方式，只有当用户真正要看这块内容的时候才去拉取。
- 7 单独合并首屏素材，用CSS控制（图片地图），首屏请求最少原则。
- 8 统计代码优化，针对用户行为统计代码，进行去除冗余，统一放到首屏后加载。

6.4.2 内容优化

一个网站呈现在用户面前，用户直接体验到的东西，就是页面的内容。页面内容是站点想要传达给用户的信息，但是信息的传递也有高效和低效之分，内容优化的目的就是使用尽可能少的内容和更高效的方式，来传达尽可能多的信息量，产生更优秀的用户体验。以下列出了一些可操作性比较强的内容优化手段。

可缓存的AJAX

Ajax经常被提及的一个好处就是由于其从后台服务器传输信息的异步性而为用户带来反馈的即时性。但是，使用Ajax并不能保证用户不会在等待异步的JavaScript和XML响应上花费时间。在很多应用中，用户是否需要等待响应取决于Ajax如何来使用。Ajax是实时响应的，在浏览器接收到新的

数据前，旧的数据被缓存，这样能够更好的提高效率。用客户端语言来判断用户当前的可视范围，只加载用户可视范围的内容。最主要的是图片，因为文字信息，相对较小，其他多媒体内容相对占用服务器流量更多。

延迟加载

大型网站中有些场景是需要呈现大量图片，例如一个有多屏的页面，多数用户点击后不会完全看完所有内容，或已经关闭或跳转到了子页面。那么实际上，我们的页面在这一过程中是100%的加载了多个屏幕的所有内容，而且如果内容过多，浏览器状态会一直显示加载状态，给用户感觉非常不好。如果可以按需加载内容，集中加载首屏时间及用户可见区域，不但可以减少加载时间，还可以减少大量带宽成本，用户不可见区域需要用户下拉滚动条或翻页时触发加载。

预加载

预加载和后加载看起来似乎恰恰相反，但实际上预加载是为了实现另外一种目标。预加载是在浏览器空闲时请求将来可能会用到的页面内容（如图像、样式表和脚本）。使用这种方法，当用户要访问下一个页面时，页面中的内容大部分已经加载到缓存中了，因此可以大大改善访问速度。下面提供了几种预加载方法：

- 1 无条件加载，触发onload事件时，直接加载额外的页面内容。
- 2 有条件加载，根据用户的操作来有根据地判断用户下面可能去往的页面并相应的预加载页面内容。

减少DOM元素数量

一个复杂的页面意味着需要下载更多数据，同时也意味着JavaScript遍历DOM的效率越慢。比如当增加一个事件句柄时在500和5000个DOM元素中循环效果肯定是不一样的。大量的DOM元素的存在意味着页面中有可以不用移除内容只需要替换元素标签就可以精简的部分。复杂的页面结构意味着更长的下载及响应时间，更合理更高效地使用标签来架构页面，是好的前端的必备条件。

使iframe的数量最小

iframe元素可以在父文档中插入一个新的HTML文档。了解iframe的工作原理然后才能更加有效地使用它，这一点很重要。<iframe>优点是解决加载缓慢的第三方内容如图标和广告等的加载问题。缺点即使内容为空，加载也需要时间，会阻止页面加载，尽可能减少iframes。

尽早刷新输出缓冲

当用户请求一个页面时，无论如何都会花费200~500ms用于后台组织HTML文件。在这期间，浏

览器会一直空闲等待数据返回。在PHP中，你可以使用flush()方法，它允许你把已经编译的好的部分HTML响应文件先发送给浏览器，这时浏览器就可以下载文件中的内容（脚本等）而后台同时处理剩余的HTML页面。这样做的效果会在后台繁忙或者前台较空闲时更加明显。输出缓冲应用最好的一个地方就是紧跟在<head />之后，因为HTML的头部分容易生成而且头部往往包含CSS和JavaScript文件，这样浏览器就可以在后台编译剩余HTML的同时并行下载它们。

当用户进行页面请求时，服务器端需要花费200~500ms时间来拼合HTML，将写在head与body之间，释放缓冲，这样可以将文件头先发送出去，然后再发送文件内容，提高效率。

使用现代化的布局方式

早期，很多站点使用table对页面进行布局，这种情况出现的原因是当时规范还不完善，为了支持更多的浏览器，不得不使用table进行页面布局。在当前情况下，HTML规范已经十分完善，主流浏览器对规范的支持也很完美，所以应尽量将table标签布局网页重构成DIV布局。相比之下，可以节约至少40%的代码量。由于DIV代码少于table布局的网页，所以搜索引擎索引权重也优于table布局的网页。另外，在HTML5规范已经成熟的情况下，尽量使用例如<header>、<footer>等语义化的HTML标签，除了能提高页面加载性能之外，还能提高代码的可维护性。

减小HTML大小

在编写代码时，为了HTML代码结构的清晰，会使用空格或者TAB进行代码缩进，同时会使用空行和注释来保证代码的可读性。但是这些在开发时带来便利的手段会增加HTML文件的大小，从而给HTML的加载带来延迟。一个好的做法是，在代码上线前对HTML进行一次Minify操作，从而去除不必要的空格、TAB、换行符、空行和注释。Minify操作一般放到上线前的代码构建流程中，并不会带来太多的开发和维护代价。另外，在编写HTML时，尽量将长URL进行压缩，站内链接使用相对路径，站外链接也可以尽量省略URL的协议部分。

还有一个减小HTML大小的方案是使用前端模板，将重复的内容（例如列表），以基础模板的形式发送到前端，然后使用JS到后端获取以JSON为格式的数据，再使用模板引擎将数据渲染出来。使用这种“JS+JSON”的方式可以减小后端渲染HTML标签带来的带宽浪费，从而提高性能。HTML大小减小10%，服务器的QPS就能提高10%，所以针对HTML的优化还是十分值得的。

6.4.3 请求优化

HTTP请求是浏览器与服务器交互的基本方式，浏览器在加载网页时至少要向服务器发出一个HTTP请求，绝大多数情况下都要发起更多的请求。根据HttpArchive的统计，目前互联网上加载一个

网页需要发出的HTTP请求数平均在90个左右。一个网页从开始加载到用户可用，中间的绝大多数时间也都花在HTTP请求上，所以对请求的优化能够得到比较大的优化收益。

尽量减少HTTP请求次数

终端用户响应的时间中，有80%用于下载各项内容。这部分时间包括下载页面中的图像、样式表、脚本、Flash等。通过减少页面中的元素可以减少HTTP请求的次数，这是提高网页速度的关键步骤。减少页面组件的方法其实就是简化页面设计，以下几个方法可以快速减少HTTP请求。

- 1 合并文件是通过把所有的脚本放到一个文件中来减少HTTP请求的方法，如可以简单地把所有的CSS文件都放入一个样式表中，当脚本或者样式表在不同页面中使用时需要做不同的修改。
- 2 CSS Sprites是减少图像请求的有效方法。把所有的背景图像都放到一个图片文件中，然后通过CSS的background-image和background-position属性来显示图片的不同部分。
- 3 图片地图是把多张图片整合到一张图片中。虽然文件的总体大小不会改变，但是可以减少HTTP请求次数。图片地图只有在图片的所有组成部分在页面中是紧挨在一起的时候才能使用，如导航栏。
- 4 内联图像是使用data:URL scheme的方法把图像数据加载页面中。这可能会增加页面的大小，把内联图像放到样式表（可缓存）中可以减少HTTP请求同时又避免增加页面文件的大小。
- 5 HTTP请求在无缓存情况下占去了40%~60%的响应时间，最大化减少请求数和缓存对网站有重要意义。

减少DNS查找次数

DNS解析的过程一般情况下返回给定域名对应的IP地址会花费20~120ms的时间，而且在这个过程中浏览器什么都不会做直到DNS查找完毕。缓存DNS查找可以改善页面性能。大多数浏览器有独立于操作系统以外的自己的缓存。由于浏览器有自己的缓存记录，因此在一次请求后，只要在过期时间内就不会受到操作系统的影响。尽可能减少网站从外部调用资源，特别是国外的应用，例如Google统计等，这种DNS查询和请求要从国外绕一圈再回国内耗时更久。

避免跳转

跳转是使用301和302代码实现的，浏览器会把用户指向到指定的URL，301和302响应都不会被缓存，而且额外产生一次请求。有一种经常被网页开发者忽略却往往十分浪费响应时间的跳转现象。重定向要尽可能规避，与404一样，最大可能不要出现。

不要出现404错误

HTTP请求都是有时间消耗的，因此使用HTTP请求来获得一个没有用处的响应（例如404没有找到页面）是完全没有必要的，它只会降低用户体验而不会有一点好处。无意义的404页面会影响用户体验并且会消耗服务器资源。

使用GET来完成AJAX请求

当使用XMLHttpRequest时，浏览器中的POST方法是一个“两步走”的过程：首先发送文件头，然后才发送数据。因此使用GET最为恰当，因为它只需发送一个TCP包（除非你有很多Cookie）。IE中URL的最大长度为2KB，因此如果你要发送一个超过2KB的数据时就不能使用GET了。POST并不像GET那样实际发送数据。根据HTTP规范，GET意味着“获取”数据，因此当你仅仅获取数据时使用GET更加有意义，相反，发送并在服务端保存数据时使用POST。Get方法和服务器只有一次交互（发送数据），而Post要两次（发送头部再发送数据）。

favicon.ico要小而且可缓存

favicon.ico是位于服务器根目录下的一个图片文件。它是必定存在的，因为即使你不关心它是否有用，浏览器也会对它发出请求，因此最好不要返回一个404 Not Found的响应。由于是在同一台服务器上，它每被请求一次Cookie就会被发送一次。这个图片文件还会影响下载顺序，例如在IE中当你在onload中请求额外的文件时，favicon会在这些额外内容被加载前下载。因此，为了减少favicon.ico带来的弊端，需要做到：

- 1 文件尽量地小，最好小于1KB。
- 2 在适当的时候为它设置Expires文件头，可以把Expires文件头设置为未来的几个月，甚至更长。

减小请求大小

浏览器在向服务器发起HTTP请求时，会带上本域的Cookie内容，由于浏览器不知道本次请求服务端到底是否需要使用Cookie，所以只要该域名下有Cookie，浏览器就会带上。对于页面中的JS/CSS/图片等静态内容，服务器在发送它们时不需要使用Cookie，所以最好的方式是不发送Cookie给服务器。为了达到不发送Cookie的目的，可以将静态文件放到一个没有Cookie的域名下。一般情况下，应该将静态文件的请求的上行开销控制在300~400B，以便达到最小代价发送请求的目的。

对于必须发送Cookie的请求，也要尽量减小Cookie的大小。Cookie在服务器及浏览器之间的通过文件头进行交换，尽可能减小Cookie体积，设置合理的过期时间，能够很好地提高效率。

通常可以通过以下几个方式减少Cookie:

- 1 去除不必要的Cookie。
- 2 使Cookie体积尽量小以减少对用户响应的影响。
- 3 注意在适应级别的域名上设置Cookie以便使子域名不受影响。
- 4 设置合理的过期时间。较早地Expire时间和不要过早去清除Cookie, 都会改善用户的响应时间。

缓存资源

用户在访问过页面一次之后, 短时间内再次访问该页面, 大多数的情况下页面中的静态资源都不会有什么变化, 所以将这些不会发生变化的资源缓存在用户的电脑上能够大大提高用户的访问速度。在HTTP协议中有几个和缓存控制相关的HTTP头信息, 其中使用起来效果比较明显的是Cache-Control。使用Cache-Control可以将你的资源缓存在用户电脑上一段时间, 你可以指定缓存时长。另外, Expires和Cache-Control作用类似, 在浏览器都已经支持HTTP/1.1的情况下, 建议只使用Cache-Control即可。一个建议是, 将缓存的有效期设置的尽可能长, 如果资源被更新过, 就使用前端构建系统将资源的URL更改掉。对于不能使用Cache-Control的资源, 也最好设置Etag或者Last-Modified, 这样当用户再次发起请求时, 服务端可以根据Etag或者Last-Modified值来判断是否需要重新发送一遍资源, 如果不需要重新发送, 就节省了再次发送资源的时间。

请求顺序优化

一个网页是由多种资源构成的, 基本包括HTML/JS/CSS/图片等。不同的资源之间有相互影响的可能, 合理安排加载顺序能够提高加载性能。一般情况下, CSS应该放到HTML的head里, 这样浏览器就能优先加载和解析CSS, 保证在网页加载的过程中页面在视觉体验上不会太差。如果将CSS放到最后加载, 那么在CSS加载之前, 网页看起来都会不符合预期, 甚至可能有些“乱”。JS文件应该放到网页的底部加载, 因为JS文件在加载后会执行, 执行的过程中可能会阻塞其他资源的加载, 从而导致整体加载变慢。一种优化实践是, JS尽量延迟加载, 在没有用到某个JS时, 该JS就不加载, 直到需要这个JS文件时再加载。另外一种实践是使用并行加载技术来加载JS文件, 先将JS文件当作非JS文件进行加载, 这样浏览器就不会阻塞其他JS请求的加载, 待所有JS加载完毕, 再使用一定的方法去启动JS的执行。

6.4.4 CSS优化

CSS是网页的重要组成部分, 页面的外观基本上由CSS决定。当前的CSS规范已经到了CSS3版本, 这使得CSS具有了更加强大的能力, 以前只能靠图片实现的效果现在也能使用CSS实现了。除此

之外，CSS还具有3D变换支持、动画支持等高级特性。另一方面，开源的CSS框架和CSS预处理器越来越多，比如bootstrap、SASS和Less等。所以针对CSS的性能优化在前端优化中的占有重要地位。

把样式表置于顶部

把样式表放到文档的<head>内部会加快页面的下载速度。这是因为把样式表放到<head>内会使页面有步骤的加载显示。注重性能的前端服务器往往希望页面有秩序地加载。同时，也希望浏览器把已经接收到内容尽可能显示出来，让浏览者能尽早地看到网站的完整样式，这对于拥有较多内容的页面和网速较慢的用户来说特别重要。

避免使用CSS表达式

表达式的问题就在于它的计算频率要比想象的多。不仅仅是在页面显示和缩放时，就是在页面滚动、乃至移动鼠标时都会要重新计算一次。一个减少CSS表达式计算次数的方法就是使用一次性的表达式，它在第一次运行时将结果赋给指定的样式属性，并用这个属性来代替CSS表达式。如果样式属性必须在页面周期内动态地改变，使用事件句柄来代替CSS表达式是一个可行办法。如果必须使用CSS表达式，一定要记住它们要计算成千上万次并且可能会对页面的性能产生影响。CSS表达式很可怕，这个只被IE支持的东西执行时候的运算量非常大，移动一下鼠标它都要进行重新计算，但有时候为了做浏览器的兼容必须要用。

使用外部JavaScript和CSS

在实际应用中使用外部文件可以提高页面速度，因为JavaScript和CSS文件都能在浏览器中产生缓存。内置在HTML文档中的JavaScript和CSS则会在每次请求中随HTML文档重新下载。这虽然减少了HTTP请求的次数，却增加了HTML文档的大小。从另一方面来说，如果外部文件中的JavaScript和CSS被浏览器缓存，在没有增加HTTP请求次数的同时可以减少HTML文档的大小。

关键问题是外部JavaScript和CSS文件缓存的频率和请求HTML文档的次数有关。虽然有一定的难度，但是仍然有一些指标可以测量它。如果一个会话中用户会浏览你网站中的多个页面，并且这些页面中会重复使用相同的脚本和样式表，缓存外部文件就会带来更大的益处。一些较为公用的JS和CSS，可以使用外链的形式。

削减JavaScript和CSS

精简是指从去除代码不必要的字符减少文件大小从而节省下载时间。消减代码时，所有的注释、不需要的空白字符（空格、换行、tab缩进）等都要去掉。在JavaScript中，由于需要下载的文件体积变小了从而节省了响应时间。精简JavaScript中目前用到的最广泛的两个工具是JSMIn和YUI Compressor。

写JS和CSS都是有技巧的，用最少的代码实现同样的功能，减少空白，增强逻辑性，用缩写方式等等，当然也有不少工具也能够帮助实现这一点。

用<link>代替@import

前面提到CSS应该放置在顶端以利于有序加载呈现。在IE中页面底部@import和使用<link>作用是一样的，因此最好不要使用它。

避免使用滤镜

IE独有属性AlphaImageLoader用于修正7.0以下版本中显示PNG图片的半透明效果。这个滤镜的问题在于浏览器加载图片时它会终止内容的呈现并且冻结浏览器。在每一个元素（不仅仅是图片）它都会运算一次，增加了内存开支，因此它的问题是多方面的。完全避免使用AlphaImageLoader的最好方法就是使用PNG8格式来代替，这种格式能在IE中很好地工作。如果你确实需要使用AlphaImageLoader，请使用下划线_filter使之对IE7以上版本的用户无效。如果需要Alpha透明，不要使用AlphaImageLoader，它效率低下而且只对IE6及以下的版本适用，用PNG8图片。如果非要使用，加上_filter以免影响IE7+用户。

正确使用预处理器

CSS是一个简单的表述编程语言。预处理器的引入突破了很多局限性，补充了急需的语言特性比如继承、混入、变量以及helper。开发人员对这些功能望穿秋水，现在像SASS、Less和Stylus这样的CSS预处理器框架在大型网站上已经无处不在了。当然预处理器带来性能提高的同时往往有负面影响。比如，CSS预处理一般支持嵌套写法，但是如果过度使用了嵌套功能会导致生成的CSS代码选择器很长，会影响页面渲染性能。

使用CSS Sprite

在Sprite中水平排列你的图片，垂直排列会稍稍增加文件大小。Sprite中把颜色较近的组合在一起可以降低颜色数，理想状况是低于256色以便适用PNG8格式，便于移动不要在Sprite的图像中间留有较大空隙。这虽然不大会增加文件大小但对于用户代理来说它需要更少的内存来把图片解压为像素地图。在CSS Sprites中竖直并尽量紧凑的排列图片，尽量将颜色相似的图片排在一起，会减小图片本身的大小及提高页面图片显示速度。另外，由于每张图片会有一个色表占据一定体积，在有比较多的小图片时，合理拼接后的图片大小会小于单张图片的总大小。

使用媒体查询（Media Query）

media type(媒体类型)是CSS2中的一个非常有用的属性，通过media type我们可以对不同的设备指定特定的样式，从而实现更丰富的界面。media query(媒体查询)是对media type的一种增强，

是CSS3的重要内容之一。使用媒体查询可以根据用户的设备特征有选择性地给出不同的资源，例如在视网膜屏幕上可以给用户显示高清图片，以免用户看到图片的锯齿感，影响体验；在普通屏幕上显示普通图片，防止传输高清图片带来的带宽浪费。

6.4.5 JavaScript优化

JavaScript在网页开发中的地位越来越重要，从以前只是用JavaScript来实现“网页特效”，到现在JavaScript已经成为了网页前端交互体验的“基石”，许许多多的网络应用使用JavaScript来实现前端业务逻辑，毫不夸张地说，由于JavaScript的广泛使用，现代网络应用的前端代码复杂度已经不下于后端代码了，有些应用的前端代码复杂度甚至比后端代码的复杂度还大。在这种情况下，针对JavaScript的优化就显得非常重要，它不但关系着网页应用在加载阶段的用户体验，更加关系着网页应用在使用阶段的用户交互体验。

把脚本置于页面底部

脚本带来的问题就是它阻止了页面的平行下载。HTTP/1.1规范建议，浏览器每个主机名的并行下载内容不超过两个。如果图片放在多个主机名上，可以在每个并行下载中同时下载2个以上的文件。但是当下载脚本时，浏览器就不会同时下载其他文件了，即脚本是串行加载。网站呈现完毕后再进行功能设置，当然这些JS要在加载过程中不影响内容表现。

剔除重复脚本

在同一个页面中重复引用JavaScript文件会影响页面的性能。重复脚本会引起不必要的HTTP请求和无用的JavaScript运算，将直接降低网站性能。重复调用的代码浏览器并不会识别忽略，而是会再次运算一遍，这当然是大大的浪费。

减少DOM访问

使用JavaScript访问DOM元素比较慢，因此为了获得更多的页面，应该做到：

- 1 缓存已经访问过的有关元素。
- 2 线下更新完节点之后再将它们添加到文档树中。
- 3 避免使用JavaScript来修改页面布局，JS访问DOM是很慢的，尽量不要用JS来设置页面布局。

开发智能事件处理程序

有时候会感觉到页面反应迟钝，这是因为DOM树元素中附加了过多的事件句柄并且些事件句柄被频繁地触发。使用event delegation（事件代理）可以很好地规避。如果在一个div中有10个按

钮，你只需要在div上附加一次事件句柄就可以了，而不用去为每一个按钮增加一个句柄。

DOM树上过多的元素被加入事件句柄的话，反应效率肯定会低，YUI事件工具具有一个onAvailable方法可以帮助你灵活的设置DOM事件句柄。

通过JS预加载图片

在空闲的时候，通过JS创建一个不在视野内的图片标签，使浏览器去加载相应的图片并缓存起来。后面需要展示这张图片的时候，就可以从缓存中快速获取图片。这种方式兼容性好，而且适用于大部分场景。

代码执行优化

- 1 使用更好的数据结构，例如hash和tree。
- 2 使用utf-8，避免转码；用array来进行字符串拼接；关键代码可以考虑手写for语句而不是forEach函数；避免使用eval；使用缓存计算（memoization）。
- 3 尽量使用局部变量，将多次使用的对象成员、数组元素及跨作用域的变量用局部变量保存。
- 4 控制原型链的深度、减少对象的嵌套。属性在原型链上的位置越深，原型数据的追溯和遍历越慢，如location.href比window.location.href快。
- 5 with/try catch/eval都会改变作用域。with会将对象添加到作用域链的头部导致标识符解析速度变慢，try-catch中的catch也类似，因此非必要情况下不要使用。
- 6 少在函数中使用闭包，闭包导致函数执行完活动对象不会立刻销毁，同时降低标识符解析速度。
- 7 正则匹配的速度比普通字符串的操作要快很多，所以尽量使用正则来操作字符串。

DOM操作优化

- 1 减少DOM访问次数，使用局部变量保存会被多次访问的DOM元素。
- 2 多使用原生JS方法，如querySelectorAll()和firstElementChild()。
- 3 批量操作DOM或样式时，将操作合并，减少重绘和重排。可将元素隐藏，操作完后重新显示，或者使用文档片段，处理好之后拷贝到页面，再或者将元素拷贝到文档流之外的节点，修改拷贝，替换原始元素。
- 4 操作html集合时，缓存集合长度。
- 5 写动画时使用绝对定位。

算法及流程控制优化

- 1 避免使用for-in循环，除非遍历属性数量未知的对象，for-in会同时遍历原型属性。
- 2 循环中减少循环次数以及每次循环中的运算步骤。
- 3 if-else和switch：条件数较少时(小于3次)使用if-else，条件较多时用switch，将出现几率最多的条件放在首位。
- 4 比较相等时尽量使用全等操作符"==="，而不是"=="，因为后者会发生类型转换。
- 5 尽量用循环取代递归，递归的关键在于终止条件，浏览器有最大递归次数限制。

快速响应用户界面

- 1 每个JavaScript任务的执行不应超过100ms。
- 2 延迟执行慢、重要性低的方法；如放在domready之后、使用定时器等。
- 3 建议定时器的最小值15ms以上，因为Windows系统定时器的精度是15.6ms。
- 4 在任务可以异步处理的情况下，拆分运行时间较长的方法。

Ajax优化

- 1 根据特定场合选择合适的数据格式。纯文本或html，CPU消耗最少，但数据量大，只适用于特定场合；XML应用广泛，但是比较笨重且解析较慢；Json轻量级，解析速度快；字符分割，轻量级，但需要服务器和客户端都要进行特殊处理。
- 2 选择合适的数据传输方式。XHR灵活性及可控性较好，但需要对数据进行处理，且不能跨域使用；动态脚本注入可以跨域，数据不需要解析，但可控性不强，不能读取响应信息。
- 3 加速Ajax。减少请求,合并文件；非重要内容放到最后请求Ajax；缓存Ajax数据。

6.4.6 图片优化

一图胜千言，可以说图片占据了网页内容的“半壁江山”。根据HTTP Archive的统计，图片传输在平均网页传输中的占比达60%以上，如表6-12所示。把网页上的图片合理的优化，对于网站性能的提升效果是非常显著的。据统计，合理地优化图片可以直接提升10%~30%+的用户体验和减少25%左右的带宽成本，甚至更高。在影响网站性能和占用带宽的众多因素中，大图片以及众多未采取优化的图片是造成这一结果的主要原因。一些1MB以上的大图片优化之后可能只有100多KB，然而在视觉感官上并没有任何影响，加载速度有数倍的差别。这涉及计算机对图像的呈现算法，以及图

像数据的压缩存放方式，并不是绝对的占用存储空间越大的图片越清晰。

表6-12 HTTP Archive元素请求和大小统计表

Content Type	Avg # of Requests	Avg size
HTML	8	44 kB
Images	53	635 kB
JavaScript	14	189 kB
CSS	5	35 kB

不要在HTML中缩放图像

不要为了在HTML中设置长宽而使用比实际需要大的图片。如果你需要，那么你的图片（a.jpg）就应该是100×100像素而不是把一个500×500像素的图片缩小使用。图片要用多大的就用多大的，1000×1000的图片被width="100" height="100"以后，本身的KB数是不会减少的。

使用Data URL

Data URL给了我们一种很巧妙地将图片“嵌入”到HTML中的方法。跟传统的用img标记将服务器上的图片引用到页面中的方式不一样，在Data URL协议中，图片被转换成base64编码的字符串形式，并存储在URL中，冠以mime-type。使用Data URL，可以减少HTTP请求。注意：Data URL建议在图片非常小的情况下使用。

使用字体图标

早些年网页上还无法显示中文，必须以图片的形式才能展现。而现在，貌似反过来了，我们可以以字体的形式展现图片。相比较图片，字体图标有很多优势，比如：

- 1 字体图标为矢量图，可以不失真的放大缩小。（目前较出名的字体文件有Font Awesome、Icon Font）
- 2 放大字体图标不会产生更大的流量消耗。
- 3 与CSS结合，可以方便的改变图标的颜色、大小、阴影等。

固定图片尺寸

网页上需要展示的图片的尺寸一般是确定的，即图片的长宽是一定的。假如需要展现的是一张200×200的图片，而开发的时候为了方便，直接使用了一张400×400的图片，将会导致网页加速速度下降，同时造成带宽浪费。清晰度相同的两张图片，传输大小会随着几何大小的增加而增加。而在普通的显示器上，用户并不会感到缩放过的大图更加清晰。

响应式图片

上一条中之所以强调普通的显示器，是因为在Retina屏出现后，对技术提出了新的要求。相对于普通的显示屏上的一个像素点，Retina屏需要绘制4个像素点。为了让一张几何大小为200×200的图片在Retina屏上更清晰的展现，需要使用实际大小为400×400的图片。但是随着Surface Pro 3的发布，高清显示屏所占份额会越来越高。如果直接载入分辨率较高的图片，可以方便地在两种屏幕上显示清晰的图片。但是这样对于非Retina屏，不仅浪费流量，还降低图片加速速度，损害用户体验。因为，推荐的做法是使用响应式图片。

图片预加载

对于一些关键的图片，例如对于图片的加载时间要求非常高，可以采用预加载的方式。

图片延迟加载

对于一些比较复杂的网站，尤其是电商网站，通常会包含大量的图片。其中一部分图片，在网页加载时不会出现在首屏中，即用户的视野中。为了提升用户体验，我们应当把先加载出现在用户视野中的图片，后加载其他图片。甚至当用户没有意愿去查看其他图片时，我们不应该加载这些图片。

但是浏览器没有那么聪明，任何出现在HTML中的图片（img标签或者某个元素的css包含图片背景），浏览器都会去加载。最糟糕的是，浏览器是按照HTML文档的顺序进行加载的，而你将一些隐藏的图片写在前面，导致首屏时间大大提高，用户觉得网站很慢。

最常见的延迟加载案例：

- 1 页面上有多个tab，仅加载当前tab中的图片。其他tab中的图片，可以使用预加载技术，或该tab被点击时再动态加载。
- 2 页面很长，下方有很多图片。通常坚挺onscroll事件，当用户即将浏览到该图片的时候进行加载。

图片自动优化平台

图片文件中含有许多冗余信息，例如空间冗余、视觉冗余等。将这些冗余信息压缩，可以减小图片的体积。基本原则是在不改变图片视觉效果的前提下将图片压缩至最小。目前有规模的团队都有建立图片优化服务，提供分辨率、质量、锐化、编码及图片适配屏幕大小、网络制式、CDN加速等功能。只要调用API和SDK即可以返回优化后的图片URL，通过分辨率、质量、锐化程度、编码格式等参数自定义调用，支持WebP、PNG、JPEG、BPG等主流格式。特别是WebP，来自腾讯的数据是减少52%，来自淘宝的数据为减少30%，来自携程的数据为减少30%~40%，各类图片格式特性如下表6-13所示。另外，图片经过优化后并推送到最优的网络环境才能达到最快的访问速度，如图6-48所示。

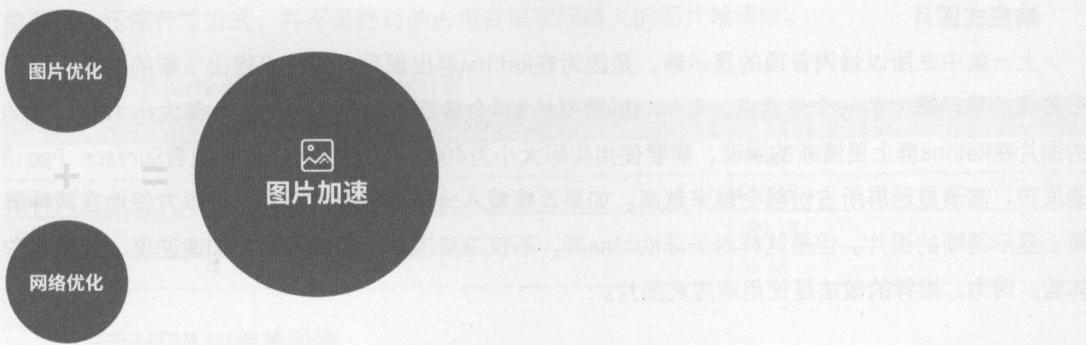


图6-48 图片优化平台示意图

表6-13 各类图片格式特点对比表

图片类型	透明	动画	浏览器兼容	适应场景
PNG	支持	不支持	所有	透明图片，简单图片
JPEG	不支持	不支持	所有	复杂颜色及形状，尤其是照片
GIF	支持（不支持半透明）	支持	所有	简单颜色，动画
WebP	支持	支持	Chrome，Opera	复杂颜色及形状，浏览器平台可预知
SVG	支持	支持	所有	简单图形，需要动态控制图片，矢量图

除了调用API和SDK在研发过程中对图片进行优化外，还可以更多个性化的线上图片优化工具进行在线对网页中的所有图片进行分析和优化，上传图片包进行批量优化，离线批量图片优化等，如图6-49所示。另外，图片优化是减少带宽成本的有效途径，图片经过自动优化后，大小会发生较大的变化，在百度经过2500万张广告图片优化得到的数据是平均减少25%左右，带宽成本与大小基本成正比例，成本对应也会减少25%左右。



图6-49 在线图片优化示意图

6.5 后端优化

与网络、系统、前端性能优化相比，后端优化因复杂、深度等原因，一般会放在最后进行优化，往往优化周期较长。前端和客户端主要承载产品功能和商业价值，而所有的逻辑、计算、存储及我们所有要解决的核心问题，扩展性、伸缩性、稳定性、性能等，其实都集中在后端。

6.5.1 架构优化

架构并非一成不变，甚至在每一个阶段都会有翻天覆地的变化。虽然新技术层出不穷，过去数年时间里，我们经历了许多激动人心的新技术，包括那些新的框架、语言、平台、编程模型等。这些新技术极大地改善了开发人员的工作环境，缩短了产品和项目的面世时间。然而无论技术怎么演变，这些都是业务架构的基本组成部分，而且一定意义上讲，架构基本决定了后端应用的性能，所以后端性能优化，架构是必须考虑的重要因素之一。

基本原则

- 1 分层低耦合的架构设计，对业务功能进行了模块化、服务化和组件化。
- 2 模块冗余，动态负载均衡，故障隔离，动态切换。
- 3 大量使用KV缓存，分布式缓存数据。
- 4 数据读写分离、动静分离。动态资源静态化，让动态数据尽可能的轻、小。（静态化是性能提升的重要途径）
- 5 对数据库进行分表、分片。
- 6 跨机房延迟优化，读本地，异步写全局。
- 7 串行改并行，同步改异步。
- 8 多运营商就近接入，例如电信、联通、移动、教育、BGP等。动态CDN加速。
- 9 谨防内存泄露，避免内存频繁申请和释放(内存池)，尽量减少内存拷贝。

微服务

微服务架构模式是近两年在软件架构模式领域里出现的一个新名词。思路不是开发一个巨大的单体式的应用，而是将应用分解为小的、互相连接的微服务，彼此之间使用统一的接口来进行交流。在功能不变的情况下，应用被分解为多个可管理的分支或服务，微服务架构模式给采用单体式

编码方式很难实现的功能提供了模块化的解决方案。由此，单个服务很容易开发、理解和维护，而且，更容易将每一个微服务的性能优化到极致。

6.5.2 并行优化

计算机的处理能力成指数能力增长，处理能力也越来越快，由于处理器的处理速度已经达到了极限，所以处理器开始向多核方向发展，而提高程序性能的一个最简单的方式之一就是充分利用多核处理器的计算资源。CPU计算受限的应用程序通过并行，可以利用所有已有的CPU核心或者升级可以提升若干个数量级的性能。跨多线程的并行工作可以更好地利用系统的资源，具有多CPU和GPU的现代计算机，通过并行可以指数级的提高CPU计算受限的应用程序的性能。除了模块内的并行，还可以考虑模块间并行。

6.5.3 异步优化

对于I/O密集型操作来说，异步执行对于应用程序的性能和伸缩性有非常关键的影响。正确使用异步编程能够使用尽可能少的线程来执行大量的I/O密集型操作。异步操作与同步是对立的，异步伴随多线程或者多进程，不但充分利用服务端并行处理能力，提高设备使用率，还提升了程序运行效率和性能。当然异步模型在编程上门槛稍高，而且会让系统更复杂。特别是I/O受限的应用程序可以将I/O操作移到另一个线程，通过执行异步的I/O操作可以提高应用程序的响应性，并很容易扩展。

6.5.4 基础优化

基础环境包括操作系统，编译器，新硬件，基础库，中间件等。基础环境调优的成本要相对较高，并且工作量很大，如从Windows平台迁移到Linux平台、从MySQL切换到NoSQL等。这类调优见效快，但受制于应用历史包袱和预算、硬件本身的限制。另外，最简单且最省事的调优方法是优化硬件资源，使用快速计算资源代替慢速计算资源，提升资源的计算能力。

- 1 操作系统升级，例如Linux 32b升级到64b及操作系统参数调优、内核升级。
- 2 第三方应用优化，Apache迁移Nginx，JBoss迁移Tomcat及版本、参数配置优化。
- 3 编译器和基础库优化，gcc、glibc升级到新版本。老的线程库效率很低，要及时升级。使用Eigen3等高性能基础数学库。
- 4 语言运行环境优化，Perl、PHP、Python、Java和Ruby等升级到高版本，同时对运行环境加速，例如PHP使用opcache、APC加速。

- 5 更快的压缩算法，例如Bzip、LZO、Snappy。
- 6 硬件更新、升级加速。
 - 1) 更快的CPU，更换高主频机器，降低处理耗时。
 - 2) 更快的本地IO设备，比如内存代替硬盘，SSD代替机械硬盘。
 - 3) 快的网络IO设备，比如使用光纤及专线减少跨区域数据中心交互延时，使用万兆网卡代替千兆、百兆网卡。
 - 4) 快速计算资源代替慢速计算资源，比如快速存储代替慢速存储，针对不同的文件选择文件系统等。
 - 5) 新硬件匹配新软件，硬件更新的同时，需要将对应的基础软件进行升级。

6.5.5 算法优化

算法非常重要，好的算法肯定会带来更好的性能。算法优化也是后端性能优化的重要内容，优化算法可以有效地提高特定场景的性能。使用一种算法时应该了解算法的适用情况、最好情况和最坏情况。如果业务逻辑中涉及一定复杂度的运算，那就可以考虑从算法和数据结构层面进行优化。如减少单台服务器（或单位计算资源）的处理量、充分利用系统资源、减少不必要的计算、减少不必要的IO等。

- 1 权衡策略和性能，权衡CPU、内存、磁盘代价，减少重复计算、Cache计算结果。
- 2 根据不同的应用场景设计不同的算法和数据结构。如果业务逻辑中涉及一定复杂度的运算，那就可以考虑从算法和数据结构层面进行优化。
- 3 寻找合适的hash算法。不同的数据不同的hash算法。
- 4 精简代码逻辑，去除冗余代码，诸如过度设计、检查等代码。精简日志操作，日志大小要关注，注意IO上的瓶颈。
- 5 去重复运算，业务逻辑优化，或者使用缓存。
- 6 能离线处理数据就不要在线处理。例如事先将数据排好序，在查询的时候采用折半或是其他查找算法以提高效率。

6.5.6 程序优化

应用程序的优化其实是整个后端优化核心，99%的性能消耗是由于1%的代码造成的。大部分性

能优化都是针对这1%的瓶颈代码进行的，找到那1%的代码，就可以优化99%的性能。如果一个应用程序存在BUG，那么即使所有其他方面都达到了最优状态，整个应用系统还是性能低下，所以，对应用程序的优化是性能优化过程的重中之重，这就对程序架构设计人员和程序开发人员提出了更高的要求。不同语言的运行速度会相当不一样，但是运行速度和开发效率是需要相互权衡的。一般根据业务需求、团队喜好来选择，Java、Python、Node.JS、Go、PHP、Ruby都不错。选定后，可以采用某些措施加速代码的运行，比如使用更好的编译器/解释器、使用C/C++编译的第三方库，等等。

6.5.7 缓存优化

缓存是减少不必要计算和I/O的重要手段，主要根据资源变化频率对资源进行分类缓存，比如动静分离和缓存等。其前提是恰当的状态管理、分离无状态的逻辑和有状态的逻辑，但会付出对一致性的一定妥协和运维的复杂为代价。缓存的适用场景包括热点不均衡、有效时间不太短、一致性牺牲程度可接受。以上所有优化手段可以组合使用，有冲突时再做权衡。

- 1 缓存Session，减少访问数据库的频率。
- 2 数据库读、写缓存，没有必要太频繁地写数据库，可以将写操作直接写在缓存上，然后累积到一定次数再一起写入数据库中。
- 3 缓存复杂运算结果，比如把计算得出的当前热点数据缓存起来。
- 4 将需要处理的数据尽量集中，可以最大限度地发挥缓存的作用。
- 5 合理的采用应用Cache，根据系统的特点选择动态Cache还是静态Cache。
- 6 后端内容预查询，提前做Cache预充。基于历史规律并利用线上空余资源主动模拟发起预取。
- 7 避免重复计算，避免不必要的网络加载。

6.6 移动优化

移动互联网用户对产品体验的容忍度很低。据权威数据分析，85%的用户希望在移动设备上加载速度至少能和在家里的计算机上一样快，几乎半数用户认为太差的性能会让他们不会再来，三分之一的用户下次就会转投竞争对手。而实际情况是移动互联网在用户行为、系统硬件和应用软件上都与传统互联网有很大的差异，加上移动产品逐渐承担了企业主要的商业价值，随着不断迭代，内容和产品逻辑越来越丰富，卡顿、闪退、不流畅、响应迟缓等用户体验问题也越来越突出。所以，移动性能优化迫在眉睫。

6.6.1 网络优化

移动网络架构、通讯机制等都跟传统网络有本质的区别，这为移动互联网的产品体验带来了很多挑战，也是移动互联网被公认的一个大问题。这些通讯机制，同时加上移动网络冗长的链路，高丢包率、高延迟，让为移动互联网的产品提供稳定、可预期的服务质量，成为非常大的挑战。

网络类型分布

如图6-50所示，WiFi用户占比已超过60%，4G用户量正接近3G用户量，2G用户在逐步减少，用户的网络越来越好。4G/3G/2G网络的带宽和延迟差别很大，2G网络上无线部分数据传输的延迟有几百ms，4G网络上无线部分传输延迟减少到几十ms，骨干网上的延迟又跟物理距离以及运营商互联互通质量有关，跨运营商50~400ms，同运营商5~80ms，这个还要取决于网络拥塞的情况。另外，在不同时间段的波动也非常巨大。

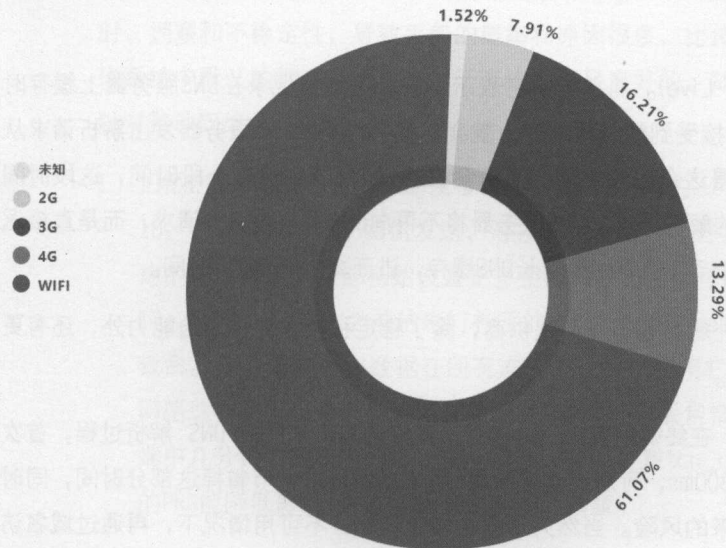


图6-50 移动用户网络接入统计

DNS优化

DNS对移动互联网的影响其实比我们想象的要大很多，如终端DNS解析滥用、域名劫持、DNS污染、老化、脆弱等。不过对于这些问题，桌面的容错性会比较好，而在手机上则比较难以解决。另一个常见问题就是DNS解析慢或者失败，例如国内中国运营商网络的DNS就很慢，一次DNS查询的耗时甚至都能赶上一次连接的耗时，如图6-51所示，尤其2G网络情况下，DNS解析失败是很常见的。因此，通过移动DNS优化，对于首次网络服务请求耗时和整体服务成功率都有非常大的影响。

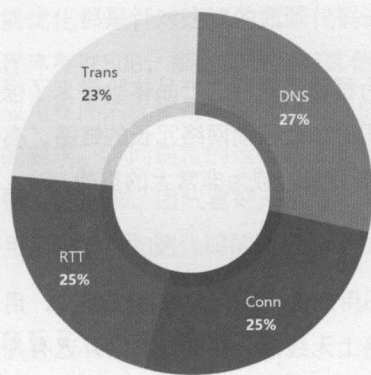


图6-51 一次网络请求耗时占比

对于DNS的问题，主要有以下几个解决思路：

- 1 减少DNS的请求、查询、更新。域名收敛，减少页面中域名的个数，如果条件允许，尽量避免用主域名。
- 2 增加DNS的TTL(Time-To-Live)。简单地说它表示一条域名解析记录在DNS服务器上缓存时间。当各地的DNS服务器接受到解析请求时，就会向域名指定的DNS服务器发出解析请求从而获得解析记录；在获得这个记录之后，记录会在DNS服务器中保存一段时间，这段时间内如果再接到这个域名的解析请求，DNS服务器将不再向DNS服务器发出请求，而是直接返回刚才获得的记录。增加 TTL值有助于延长DNS缓存，进而减少DNS解析时间。
- 3 选用更优质的国内、国外域名解析服务提供商，除了稳定和较大的抗风险能力外，还有更多的DNS缓存服务器可用。
- 4 不用域名，用 IP 直连。在终端配置server list，直接访问IP，省去 DNS 解析过程，首次域名解析一般需要200~300ms，可通过直接向IP 而非域名请求，节省掉这部分时间，同时可以预防域名劫持等带来的风险。当然为了安全考虑，当IP不可用情况下，再通过域名访问。
- 5 TDO (Traffic Distribution Optimizer) 优化。可以理解是一种用户定向策略，DNS以TDO为依据，TDO首要目标是规避跨网解析。在移动领域，针对海量移动用户测速，根据用户请求来源IP段，从而判断该IP段用户的访问落到其接入速度最快的服务器，这个过程适应域名解析和IP直连。
- 6 HttpDNS优化。与传统走UDP协议的DNS不同，HttpDNS基于HTTP协议，绕过Local DNS，从而减少域名解析的时间并解决DNS劫持的问题。HttpDNS能够得到客户端的出口网关IP，从而

能够更准确地判断客户端的地区和运营商，得到更精准的解析结果。而且HttpDNS客户端SDK有几个特性：预解析、多域名解析、TTL缓存和异步请求都可以减少DNS时间。

接入优化

全国多运营商分布式部署，除了移动、电信、联通主流运营商实现全国大区分布外，每一个主流运营商默认不能识别的用户需要解析到北、上、广多线BGP，全国重点部署小运营商多线BGP覆盖。关于动态应用，一二线城市可以考虑动态代理或动态CDN加速，配合上面说到的动态移动TDO解析策略，每次根据地域、网络类型等选择最优的服务器进行连接。

协议优化

- 1 TCP拥塞窗口大小。TCP拥塞控制是一个看门人，决定如何控制从您的服务器到客户端的数据包流量。其目的是为了防止网络拥塞，当拥塞发生时检测并减慢数据的传送速度。这有助于确保互联网可用，但可能会导致移动网络上的问题。而移动无线网络本身具有高延迟、拥塞和不稳定性，导致丢包的概率和原因很多，比传有线网络要高出很多。在很多操作系统中默认的拥塞控制算法对于有线网络是良好的，却在移动网络有问题，所有，需要针对移动网络的TCP优化。
 - 1) 旧的Linux内核不适合移动设备访问。因为Linux的2.6.38默认initcwnd和initrwnd是3和10，它允许14.2KB初始值发送，等接收的返回后才开始缓慢增长。这是为HTTP和SSL考虑的，因为需要头部初始设置更多空间的数据包。可以将initcwnd和initrwnd值设置为10。另外，3.2版本之前的未打补丁的内核tcp_no_metrics_save = 0 的sysctl设置会更致命。此设置将保存数据在所有连接上，并试图用它来优化网络。不幸的是，在移动网络这样实际上使性能更差，因为TCP会将一个丢包情况下采取的策略适用于这个客户端中几分钟的窗口内的每一个新连接。可以设置tcp_no_metrics_save = 1 ,Linux的3.2的PRR能降低影响TCP性能的有损连接的数量。
 - 2) Linux3.x版本。内核中的一个重要的设置: tcp_slow_start_after_idle，默认情况下，此参数设置为1，将大幅削减空闲连接时的CWnd，对长连接会产生负面影响，将其设置为0，可以显著提高长连接性能。
 - 3) 多个TCP拥塞控制算法中，在大量研究的基础上发现tcpwestwood对于无线网络是一种较理想的算法。它的主要思想是通过在发送端持续不断的检测ack的到达速率来进行带宽估计，当拥塞发生时用带宽估计值来调整拥塞窗口和慢启动阈值，从而提高了无线网络的吞吐量。

- 2 TF0 (tcp fast open)，即TCP_FASTOPEN参数，这个是Google提交的一个rfc，是对TCP的一个增强，客户端通过TCP连接到服务器时，可以在SYN报文携带数据，简而言之就是在3次握手的时候也用来交换数据。
- 3 网络服务重发机制，移动网络不稳定，如果一次网络服务失败，就立刻反馈给用户你失败了，体验非常不友好。可以提供了网络服务重发机制，即当网络服务在连接失败、写Request失败、读Response失败时自动重发服务。长连接失败时就用短连接来做重发补偿，短连接服务失败时当然还是用短连接来补偿，这种机制增加了用户体验到的服务成功概率。
- 4 SPDY和HTTP 2.0协议的目标都是通过更有效地利用底层的TCP连接（多路复用、头压缩、优先化处理），来减少页面的加载时间，而且服务器push通过消除额外的网络延迟，可以进一步提高网页性能。

6.6.2 请求优化

在互联网的世界里，一个简单的网页请求过程其实是非常复杂的。Web基本协议是HTTP协议，它跑在TCP协议之上，而TCP协议又需要IP协议的支持，IP协议又要由底层链路来支撑，每一个网页请求都要经历HTTP→TCP→IP→链路层协议等复杂过程。而每一个页面加载都需要发生多个网页元素构成请求，所以这些请求基本决定了网页加载的效率和速度，请求优化自然也成为性能优化的重要内容，特别是移动优化。

连接复用

通俗来讲，浏览器和服务器每进行一次通信，就建立一次连接，任务结束就中断连接，即短连接。相反地，假如通信结束后保持连接则为长连接。从HTTP/1.1起，默认使用长连接，这样做的优点是显而易见的，一个网页的加载可能需要HTML文件和多个CSS、JS、图片等，假如每获取一个静态文件都建立一次连接，那么就太浪费时间了。对于频繁请求资源的客户来说，较适用长连接。但连接数最好进行限制，防止建立太多连接拖累服务端。长连接也不能无限期地长，移动设备上保持长连接较耗电，需要通过在线状态等特征判断结束长连接，服务端需要灵活设置Keep-Alive，其中timeout等于一个值来规定保持连接的秒数，还可以用max来规定多少次请求后断开。

另外，HTTP1.1协议规定请求只能串行发送，这也是HTTP性能最让人诟病的地方。SPDY和HTTP2核心优势就是多路复用，简单说来就是将多个请求通过一个TCP连接发送。

请求合并

即将多个请求合并为一个进行请求，在移动端，打包预备加载，主动推送到客户端等很多灵活的方式将多个URL请求变成了一个请求。

减小请求数据大小

- 1 对于 POST 请求, Body 可以做 Gzip 压缩, 如日志。
- 2 对请求头进行压缩。这个Http1.1不支持, SPDY及Http2.0支持。Http1.1可以通过服务端对前一个请求的请求头进行缓存。
- 3 控制请求头大小。请求头部瘦身, 例如控制Cookie大小等。

减小返回数据大小

- 1 压缩。例如文本、API 数据使用 Gzip 压缩。
- 2 精简数据格式。例如JSON 代替 XML。
- 3 增量更新。需要数据更新时, 只进行增量更新。
- 4 断点续传。Android和iOS都支持断点续传, 特别是大文件。
- 5 图片优化。移动端图片尽可能使用webp, 来自BAT的数据显示, 相比优化前提速30%~50%, 同时支持图片适配。小图可以考虑转换成base64编码的字符串形式使用。

异步加载

移动互联网中, 尽可能早地渲染页面, 让白屏时间更短, 让用户提早看到首屏和可操作界面, 一直是性能优化的重要内容。异步和并行加载是主要的实现方式, 目前实现异步和并行加载方式较多, 就不在这里详细介绍。

6.6.3 缓存优化

浏览器花费了绝大部分时间去获取包括脚本、样式和图片在内的一些组件, 缓存直接减少HTTP请求的次数对于减少加载时间有着重大的影响。移动缓存问题不能简单理解为元素加载, 而是要从产品整个生命周期来理解。浏览器缓存以URL为单位, 一个URL可能对应多个文件的打包, 多个文件合并成一个URL, 随着合并文件的增多, 每次上线URL缓存失效的可能性会非常高。所以, 非常有必要针对移动缓存进行深度优化。

预加载优化

预加载是预置一份标准缓存到application cache的缓存目录, 解决首次加载问题, 之后通过增量更新不断迭代。用户访问之前, 将页面静态资源 (HTML/JS/CSS/IMG...) 打包预加载到客户端, 用户访问时, 将网络IO拦截并替换为本地文件IO, 从而实现加载时间的大幅度提升。因为Android和iOS安装

包已经很大，所以预加载Zip包需要从服务器端下载到客户端，本地需要记录整体包状态，并在合适的时机（网络通畅、资源闲置等）与服务器通信并交换状态信息。在包发布更新的过程中要注意本地版本和服务端最新包之间的境量同步，需要进行必要的网络判断，最好在WiFi下才下载。

数据访问量较大，并且在服务器资源很有限的情况下，需要借助CDN来分流。需要注意的是预加载实际上也是一种缓存，主要受推送到达率（用户是否在线，用户所在网络质量）的影响，所以需要推、拉结合的触发策略，并优化下载包的体积（压缩增量包）来提升传输效率和到达率。

本地缓存

虽然可以通过浏览器缓存静态文件，用户主动触发的页面刷新行为（比如刷新按钮、右键刷新等），会导致浏览器放弃本地缓存，还是会发起 `cache-control:max-age=0` 的请求，如果没有新内容，返回304。如果使用localStorage，可以完全避免这种情况，等效于无视用户主动刷新行为的本地强缓存。而且，多数情况下localStorage快于304。

同时，使用localStorage进行本地缓存，通过缓存页面框架及页面元素，大幅提升传输效率。腾讯基于localStorage的缓存控制最为细致，可以做到字符级别的资源增量更新。在快速迭代版本过程中，有时候只修改了某些文件中的几行代码，却需要用户下载整个JS文件，这在移动端显得非常浪费，腾讯基于localStorage的增强更新算法实现了修改多少代码就只下载修改代码的功能，让版本迭代过程中缓存的命中率大大提高。最大限度减少了更新内容，同时也为用户和企业节省大量流量。

6.6.4 策略优化

在无线网络整体容错能力较小的大环境下，从监测数据中发现慢速网络下用户访问的速度及成功率都较差，而且慢速用户不是相对固定的，而是相对分散和流动的。这些与有线网络有本质的差别，也决定了解决移动访问成功率及慢访问速度的问题不能像PC端一样，只要通过分布式的多运营商和多IDC覆盖就能有较大改善。一定意义上，这种特殊的移动网络和移动终端造成的慢网速是不可抗的，所以，催生了通过改变这两者之间的传输内容来实现更好体验的优化方向。这也是腾讯的产品文化之一，柔性可用。

实现原理

移动柔性可用与PC不一样，PC的端和网络都相对较快，可以通过前端判断网速，并且在事中快速响应，而移动端不行。移动端只能在事前基于大规模测速数据，建立慢速用户库，也就是来源IP段与其平均访问速度的对应关系，然后将快、慢网速用户解析到对应的访问内容。慢网速用户访问

到简单版，而快网用户访问到丰富版本，而且两种版本提供自选切换选项，由用户选择，即使从轻版本切换丰富版，用户也愿意等待。一些细节考虑：

- 1 如用户从2G慢速网络切换到WiFi快速网络，切换网络后其原IP会改变，进而访问到的内容也会发生变化。
- 2 当用户在简单版与丰富版之间切换时，会在Cookie中记录其选择，在下次请求时便不再主动为其切换版本。
- 3 简单版与丰富版之间不是绝对隔离的，简单版可以将功能下沉，供用户选择，例如快网速用户直接看到高清大图，慢网速用户看的是小图，是可以点击下载原图来更细粒度的增加体验。
- 4 更细致化的定制，对于不同的厂商、设备、硬件返回不同的内容，例如针对不同分辨率，显示不同压缩比的图片等。

优化收益

大幅提升慢网用户访问速度和成功率，同时减少用户带宽费用。根据经验数据，速度提升40%~80%的同时，还会增加4%~8%的访问量，而且90%以上的慢速用户选择留在简版。

其他途径

设计得好的应用，即使底层连接慢或者请求时间很长，通用UI中提供即时反馈也能让人觉得速度快。有专业人士做过一个实验，有一个请求需要花2m，有一半的人只等了8.5s就离开了，通过增加loading之后离开的时间延到了20s，而有进度条则让多数用户等到了最后。腾讯也尝试在较长时间等待的场景下，先让用户看或玩一些娱乐性的动画。我们经常能看到，一些站点在首次进入的时候会先显示一个进度条，如图6-52所示，等资源加载完毕后再呈现页面，然后整个页面的操作就会非常流畅，因为之后没必要再等待加载资源了。尤其是在移动端，或者是页游中，这样做能避免页面出现白屏，很大程度提升用户体验。

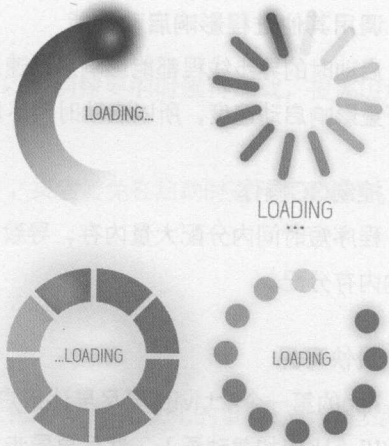


图6-52 进度条示意图

6.6.5 启动优化

Native App的启动速度、使用流畅度是移动用户体验的重要性能指标之一，开发时要把启动速度、操作流畅度作为重要关注点。

初始化Layout要简洁

暂时不使用的UI元素不要放到初始化Layout中，减少Layout复杂度，加速启动速度。这些暂时不使用的UI可以使用stubview概念，采用lazyload的方式，使用时再加载，分散CPU消耗。

延后加载

App初始化时只需要初始化必要的资源，尽量减少初始化暂时不需要的资源，减少启动时的CPU消耗。这些资源可以使用时再进行初始化。

异步线程

App启动时难免使用异步线程来执行一些任务，比如联网激活，通过网络接口获取最新数据等。CPU的运算次数是有限的，异步线程反应在CPU上也只是CPU的分时操作，如果异步线程操作占用CPU过多时间，则启动时的UI线程占用的CPU就会减少，则影响启动速度。因为异步线程能够影响启动性能，所以我们应该尽量控制异步线程的数量以及其所执行的操作。必要时可以降低异步线程的优先级，尽量减少对UI线程的影响。

调用其他进程影响启动性能

启动时的异步线程都能影响启动速度，异步调用其他进程也是同样道理。进程启动开销更大，会严重影响启动速度，所以启动时要尽量不要和其他进程有异步的交互。

控制GC操作

程序短时间内分配大量内存，导致垃圾回收操作，会严重影响启动性能。所以要严格控制启动时的内存分配。

最快可见

App的第一个Activity要尽早让用户看见一些UI元素，避免白屏很长时间。先让用户看见一个初始化UI（比如加载动画），然后再异步加载数据，更新UI。

6.6.6 交互优化

App的交互性能直接体现了应用的流畅程度和用户体验，主要体现在页面切换、图形和动画展

示、页面滑动效果等。尽管移动开发框架功能和性能都非常强大，但如果对其原理不清楚也会降低效率。让App的交互性能达到最优的状态是一门艺术。App慢交互的直接体现为卡顿。在开发时，可以从以下几个方面优化APP的交互性能：

- 1 优化布局，尽量减少视图层次。当视图多个层次重叠在一起显示时，GPU需要不停地重绘同一区域，引起掉帧现象。
- 2 慎用Alpha。
- 3 尽量减少在短时间内大量图片的显示，尽可能将多张图片合成为一张进行显示。
- 4 尽量避免调整视图层次、添加和移除视图。
- 5 尽量提前计算好布局，在需要时一次性调整好对应属性，而不要多次、频繁地计算和调整这些属性。
- 6 避免临时转换。确保图片大小和frame一致，尽量避免在滑动时缩放图片，确保图片颜色格式被GPU支持，避免劳烦CPU转换。
- 7 对于较复杂的、静态的效果等，预先渲染成位图(bitmap)，然后加入缓存中。如对于阴影效果这样比较消耗资源的静态内容进行缓存，可以得到一定幅度的性能提升。
- 8 避免在 UI 线程进行繁重的操作。耗资源的操作（比如 IO 操作、网络操作、SQL 操作、列表刷新等）耗资源的操作应用后台进程去实现，不能占用 UI 线程，UI 线程是主线程，主线程是保持程序流畅的关键。
- 9 按需载入视图。某些不怎么重用的耗资源视图，可以等到需要的时候再加载，提高UI渲染速度。
- 10 优化应用的启动速度，可以在启动时延迟加载一些UI，或者避免在启动时进行繁重操作。

6.6.7 内存优化

iOS内存优化

在iOS开发的过程中，性能优化是永恒的话题。而内存优化是iOS性能优化的重要部分，也是iOS开发的核心关注点之一。iOS可以从以下几个方面来优化APP的内存。

- 1 降低内存占用。iOS使用的是低内存处理机制Jetsam（也可以称作Memorystatus），概念上类似于Linux的“Out-Of-Memory” killer（俗称OOM），最初是用来Kill消耗太多内存的进程。当内存过低的时候，就会在队列中进行广播，希望大家尽量释放内存，如果一段时

间后，仍然内存不够，就会开始Kill进程。因此，可以通过降低内存峰值，降低APP的内存占用，来优化APP性能。

- 2 AutoReleasePool（自动释放池）。在MRC模式下，开发者需要手动的调用Retain和Release方法来管理对象的引用计数。那么可能在一个线程中有大量的Retain和Release方法，这种情况下，使用自动释放池统一管理会方便很多。将线程中要执行的任务都放在自动释放池中，自动释放池会捕获所有任务中的对象，在任务结束或线程关闭之时自动释放这些对象。
- 3 缓存处理。缓存最适合存储频繁访问的对象，尽管 NSCache 和其他缓存库在内存不足时会丢弃部分缓存，但开发人员需要更多控制权。比如在应用程序中，我们下载了大量不同图像并缓存它们，而APP也没有频繁访问这些图像。那么很可能随着APP运行，导致内存不必要地不断增长。

因此，作为开发人员，需要判断使用缓存系统是否有好处，是否需要加载的资源缓存。

- 4 处理内存警告。上文有提到“当内存过低的时候，就会在队列中进行广播，希望大家尽量释放内存……”，在实际应用过程中，一旦系统内存过低，iOS会通知所有运行中App。若不处理，APP就可能被系统Kill。因此，需要对此类内存警告及时处理。

在iOS开发中，可以通过app delegate中applicationDidReceiveMemoryWarning: 方法、UIApplicationDidReceiveMemoryWarningNotification通知等方式获取内存警告信号，并在收到警告后尽可能多的释放资源，尤其是图片等占用内存多的资源，等需要的时候再进行重建。

- 5 避免内存泄露。iOS内存管理是个大问题，在编程过程中，及时释放不需要的内存对象，是基本原则。对象不及时释放和过早释放等，都可能会导致程序直接crash。
- 6 使用ARC模式。自动引用计数（ARC），是一项为Objective - C程序在编译时提供自动内存管理的功能。开发者不再需要手动调用retain、release、autorelease这些方法来管理对象的引用计数，其工作原理是将内存操作的代码(retain、release等)自动添加到需要的位置，保证释放掉不再需要的对象的内存。
- 7 借助内存检测工具。即使有 ARC（自动引用计数）内存管理机制，但在现实中对象之间引用复杂，循环引用导致的内存泄漏仍然难以避免，所以关键时刻还要自力更生。iOS可以通过自带的工具排查内存泄露：

Analyze：对代码进行代码静态检查，分析哪里存在内存泄露。

Leak：可以帮助找到引发内存泄漏的起点。

Android内存优化

Android的程序由Java语言编写，所以Android的内存管理与Java的内存管理相似。程序员通过new为对象分配内存，而对象的释放是由垃圾回收器来完成的。

虽然有垃圾回收机制，但并不表示没有内存泄露。其实如果一个程序中，已经不再使用某个对象，但是因为仍然有引用指向它，垃圾回收器就无法回收它，这就造成了内存泄露。如果我们的Java运行很久，而这种内存泄露不断地发生，就会导致严重的性能问题。

可以从以下几个方面来进行内存优化：

1 减小对象的内存占用

- 1) 减小Bitmap对象的内存占用，如通过在把图片载入内存之前，先计算出一个合适的缩放比例，避免不必要的大图载入。
- 2) 静态方法代替虚拟方法，如果不需要访问某对象的字段，将方法设置为静态。
- 3) 尽量避免static成员变量引用资源耗费过多的实例，比如Context。
- 4) 避免内部Getters/Setters，在Android中，虚方法调用的代价比直接字段访问高昂许多。
- 5) 使用实体类比接口好，在Android中，调用一个接口的引用会比调用实体类的引用多花费一倍的时间。
- 6) 避免使用枚举Enum，枚举变量非常方便，但不幸的是它会牺牲执行的速度和大幅增加文件体积。

2 内存对象的重复利用

- 1) Bitmap图片缓存，如采用ImageCache、Picasso Square、Android-Universal-Image-Loader等框架缓存。
- 2) 优化Adapter适配器。如在Adapter中使用convertView和ViewHolder来进行缓存处理。
- 3) 利用线程池管理线程，避免反复创建线程对象所带来的性能开销，节省了系统的资源。

3 内存对象回收

- 1) 及时回收Thread（线程），线程中涉及的任何东西GC都不能回收，所以线程很容易造成内存泄露。

- 2) 及时回收Cursor。在大多数情况下,应及时关闭Cursor,保证Cursor占用的内存被及时的释放掉。在CursorAdapter中应用,不能直接将Cursor关闭,需要在onDestroy函数中,手动关闭。
- 3) 及时回收Receiver,调用registerReceiver()后续在适当时及时调用unregisterReceiver()。
- 4) 及时回收Stream/File(流/文件),如InputStream/OutputStream, SQLiteOpenHelper, SQLiteDatabase, Cursor, 文件, I/O, Bitmap图片等操作都需要显示关闭。

6.7 其他优化

除了网络、系统、前端、后端、移动端等主流性能优化方向外,还有一些前沿的优化方向需要关注,例如SPDY、HTTP/2、ESI、SDCH、BigPipe、DNS Prefetch、HHVM等。

6.7.1 SPDY

SPDY并不是首字母缩略字,而仅仅是"speedy"的缩写,SPDY是一种开放的网络传输协议,由Google开发,基于传输控制协议(TCP)的应用层协议。Google最早是在Chromium中提出的SPDY协议。目前已经被用于Google Chrome浏览器中来访问Google的SSL加密服务。SPDY当前并不是一个标准协议,但SPDY的开发组已经开始推动SPDY成为正式标准,HTTP/2主要以SPDY技术为主。Google Chrome, Mozilla Firefox, Opera和Internet Explorer均已支持SPDY协议。SPDY协议类似于HTTP,但旨在缩短网页的加载时间和提高安全性。SPDY协议通过压缩、多路复用和优先级来缩短加载时间。

SPDY协议只是在性能上对HTTP做了很大的优化,其核心思想是尽量减少连接个数,而对于HTTP的语义并没有做太大的修改。具体来说,SPDY使用了HTTP的方法和页眉,但是删除了一些头并重写了HTTP中管理连接和数据转移格式的部分,所以基本上是兼容HTTP的。

- 1 多路复用请求优化,SPDY规定在一个SPDY连接内可以有无限个并行请求,即允许多个并发HTTP请求共用一个TCP会话。这样SPDY通过复用在单个TCP连接上的多次请求,而非为每个请求单独开放连接,这样只需建立一个TCP连接就可以传送网页上所有资源,不仅可以减少消息交互往返的时间还可以避免创建新连接造成的延迟,使得TCP的效率更高。此外,SPDY的多路复用可以设置优先级,而不像传统HTTP那样严格按照先入先出一个一个处理请求,它会选择性的先传输CSS这样更重要的资源,然后再传输网站图标之类不太重要的资源,可以避免让非关键资源占用网络通道的问题,提升TCP的性能。

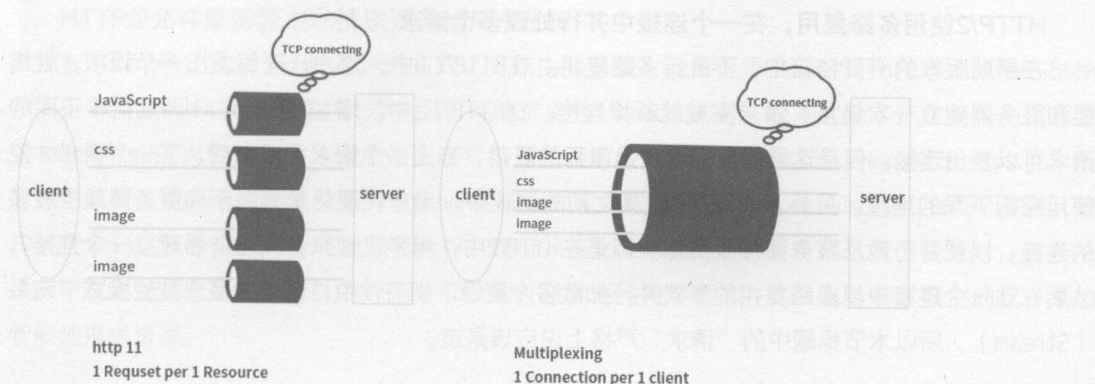


图6-53 SPDY多路复用示意图

- 2 支持服务器推送技术，服务器可以主动向客户端发起通信向客户端推送数据，这种预加载可以使用户一直保持一个快速的网络。
- 3 SPDY压缩了HTTP头，舍弃掉了不必要的头信息，经过压缩之后可以节省多余数据传输所带来的等待时间和带宽。
- 4 强制使用SSL传输协议，Google认为Web未来的发展方向必定是安全的网络连接，全部请求SSL加密后，信息传输更加安全。

6.7.2 HTTP/2

HTTP/2是基于Google的SPDY协议开发出的新一代HTTP协议。上一个HTTP协议是HTTP/1.1，也是目前在互联网上被广泛使用的HTTP协议。HTTP/2针对HTTP/1.1的不足和缺点做了较大的改进，从而提高网页加载的性能。各大浏览器厂商也在积极跟进HTTP/2，Firefox的最新版本Firefox36已经完整支持HTTP/2的所有功能，Chrome和Windows10预览版中的IE也都提供了对HTTP/2的支持。HTTP/2具有以下几个重要特性：

HTTP/2基于二进制，而非基于文本

HTTP/1.1及其以前的版本，都是基于文本的协议。文本协议的缺点在于传输时需要传输更大量的内容，同时在内容编码上出bug的概率较高，解析时相对于二进制解析来说也比较低效。HTTP/2抛弃了文本协议，采用了二进制协议，这可以让协议工作起来更简单，错误也更少。当然带来的问题也是显而易见的，让调试变得不再直观。在HTTP/1.1协议之上，调试错误只需要直接观察发送的请求和响应即可，文本协议十分便于人工调试，使用telnet就可以发起一个HTTP请求。在HTTP/2协议之下，调试错误可能就需要借助一些相对较为复杂的工具才行。

HTTP/2使用多路复用，在一个连接中并行处理多个请求

在早期版本的HTTP协议中，不进行多路复用。在HTTP/1.0中，用户代理每发出一个请求，就需要和服务器建立一次链接，请求完成就断掉连接。在HTTP/1.1中，增加了keep-alive功能，不同的请求可以复用连接。但是这里的复用仅仅是串行的复用，当上一个请求完成之后，下一个请求才能使用空闲下来的连接。另外，HTTP/1.1及其之前的协议中，用户代理总是倾向于和服务器建立较多的连接，以便并行地从服务器加载资源。但是在HTTP/2中，用户代理只会和服务器建立一个连接，然后在这一个连接中以多路复用的形式并行加载多个资源。HTTP/2中，请求的概念被变换成“流”（Stream），所以本节标题中的“请求”严格上说应该是流。

在HTTP/2中，数据的传输是以帧（Frame）为单位传输的，所有的帧都传输在同一个连接上，有一些帧具有相同的“编号”，同编号的帧被用来传输同一个资源，也就构成了一个“流”。不同的资源都在同一个连接上传输，而且是并行传输，这就是多路复用。HTTP/2协议还是一个全双工协议，在服务端向用户代理传输数据的同时，用户代理也可以向服务端传输数据。其示意图如图6-54所示。

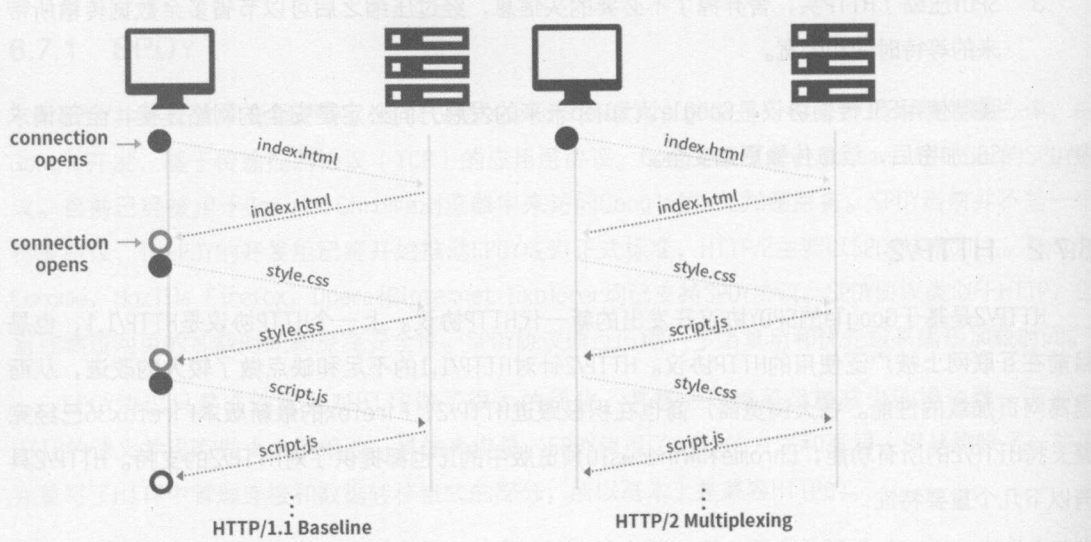


图6-54 HTTP/1.1与HTTP/2对比图

HTTP/2对头部进行压缩

在HTTP/1.1中，HTTP头部是使用文本明文传输的，例如http头，这种头信息带来的问题是，每次发起HTTP请求，都会传输大量的头信息，所以HTTP/2对头信息进行了压缩，不再使用明文表示，而是定义了HPACK规范，将头信息使用二进制的方式表示。这么做带来的好处是，建立一个新的流，最优情况下，仅需要9B即可。相比于文本方式表示的头信息，这样可以节省大量的数据传输量，从而提高效率。

HTTP/2允许服务器主动推送资源给用户代理

在HTTP/2中,定义了一种推送的方式,可以由服务端主动将资源推送给用户代理(如图6-55所示)。举例说明一下:当用户代理向服务端请求文档index.html时,服务端除了正常向用户代理发送index.html的内容外,服务端还知道该文档再后续解析过程中会使用到style.css和script.js,所以服务端就直接向用户代理发送两个promise帧,向用户代理推送style.css和script.js两个资源。用户代理拿到这两个资源后会将他们暂时存入缓存,当index.html解析到需要这两个资源时,直接从缓存中取出使用即可。这样做的好处是可以最大限度地利用带宽,并且让服务端掌握主动权,更加智能地推送资源。

Web Server可以使用如下方案来确定推送哪些资源。首先Web Server进行一个“学习”的过程,根据用户发送过来的请求中的Referer头来确定哪些资源需要推送以及在何时推送。比如,用户代理在请求style.css时,会带上值为index.html的Referer头,这样Web Server就知道style.css是index.html所需要的资源,就可以在下次用户代理请求index.html时主动将style.css推送出去。

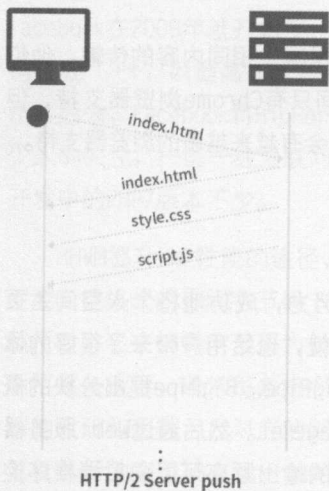


图6-55 HTTP/2服务器推送示意图

6.7.3 ESI

ESI (Edge Side Includes)是一种数据缓冲/缓存服务器,它提供将Web网页的部分(这里指页面的片段)进行缓冲/缓存的技术及服务。以往的数据缓冲服务器和信息传送服务以页为单位制作,复制到数据缓冲服务器中,由于其对应要求而进行传送,所以向网络软件等根据用户的输入,内容会动态地转变的网页传送信息的时候,就很难得到高效率。由于在ESI中是部分地缓冲网页,使用基于XML的标记语言,指示想要缓冲的页面部分。由此,页面内分为动态地变更的部分和静态的不

变更的部分(网站内的共通菜单等), 只将静态的部分有效地发送到服务器中。

通过使用简单的标记语言来对那些可以加速和不能加速的网页中的内容片断进行描述, 每个网页都被划分成不同的小部分分别赋予不同的缓存控制策略, 使Cache服务器可以根据这些策略在将完整的网页发送给用户之前将不同的小部分动态地组合在一起。通过这种控制, 可以有效地减少从服务器抓取整个页面的次数, 而只用从原服务器中提取少量的不能缓存的片断, 因此可以有效降低原服务器的负载, 同时提高用户访问的响应时间。与SSI不同的是, ESI多在缓存服务器或代理服务器上执行。

6.7.4 SDCH

SDCH是Shared Dictionary Compression over HTTP的缩写, 即通过字典压缩算法对各个页面中相同的内容进行压缩, 减少相同的内容的传输。例如一个网站中一般都是共同的头部和尾部, 甚至一些侧边栏也是共同的。之前的方式每个页面打开的时候这些共同的信息都要重新加载, 但使用SDCH压缩方式的话, 那些共同的内容只用传输一次就可以了。

SDCH压缩方式是为了减少相同内容的传输的, ajax+pushState也是减少相同内容的传输, 他们想达到的效果是一样的。只是SDCH是Google出的, 可能今后一段时间只有Chrome浏览器支持, 但pushState是HTML5的一个标准, 目前已经有Chrome和Firefox支持, 之后会有越来越多的浏览器支持。

6.7.5 BigPipe

Facebook的前端性能研究小组在一次性能优化项目中经过数月努力, 成功地将个人空间主页面加载耗时由原来的5s减少为现在的2.5s。这是一个非常了不起的成就, 也给用户带来了很好的体验。在优化项目中, 工程师提出了一种新的页面加载技术, 称之为BigPipe。BigPipe提出分块的概念, 即根据页面内容位置的不同, 将整个页面分成不同的块, 称为pagelet。然后通过Web 服务器和浏览器之间建立管道, 进行分段输出, 减少大量请求数。Pagelet的输出顺序可以由后端程序控制, 及早输出用户关心的模块。

该技术的设计者Changhao Jiang 是研究电子电路的博士, 可能从微机上得到了启发, 将众多pagelet加载的不同阶段像流水线一样在浏览器和服务器上执行, 这样就做到了浏览器和服务器的并行化, 从而达到重叠服务器端运行时间和浏览器端运行时间的目的。使用BigPipe不仅可以节省时间, 使加载的时间缩短, 而且可以同过pagelet的分步输出, 使一部分的页面内容更快地输出, 从而获得更好的用户体验。

BigPipe适用于第一个请求时间较长, 后端程序需要读取多个API的应用, 页面上的动态内容可以划分在多个区块内显示, 并且各个区块之间的关联不大。

6.7.6 DNS Prefetch

DNS Prefetch是一种DNS预解析技术，当你浏览网页时，浏览器会在加载网页时对网页中的域名进行解析缓存，这样在你单击当前网页中的连接时就无需进行DNS的解析，减少用户等待时间，提高用户体验。如果要控制浏览器端是否对域名进行预解析，可以通过Http header的x-dns-prefetch-control属性进行控制。DNS Prefetch 已经被主流浏览器支持。现在大多数新浏览器已经针对DNS解析进行了优化，典型的一次DNS解析耗费20~120ms，减少DNS解析时间和次数是个很好的优化方式。DNS Prefetching是具有此属性的域名不需要用户点击链接就在后台解析，而域名解析和内容载入是串行的网络操作，所以这个方式能减少用户的等待时间，提升用户体验。

6.7.7 HHVM

HHVM (HipHop Virtual Machine) 起源于Facebook公司，Facebook早期的很多代码是使用PHP来开发的，但是，随着业务的快速发展，PHP执行效率成为越来越明显的问题。为了优化执行效率，Facebook在2008年就开始使用HipHop，这是一种PHP执行引擎，最初是为了将Facebook的大量PHP代码转成 C++，以提高性能和节约资源。使用HipHop的PHP代码在性能上有数倍的提升，如图6-56所示。后来，Facebook将HipHop平台开源，逐渐发展为现在的HHVM。HHVM因为它的高性能而吸引了不少人的关注，一些一线互联网公司也开始跟进使用。从纯语言执行性能测试结果来看，HHVM领先了开发中的PHP7版本不少。

HHVM提升PHP性能的途径，采用的方式就是替代Zend引擎来生成和执行PHP的中间字节码（HHVM生成自己格式的中间字节码），执行时通过JIT（Just In Time，即时编译是种软件优化技术，指在运行时才会去编译字节码为机器码）转为机器码执行。Zend引擎默认做法，是先编译为opcode，然后再逐条执行，通常每条指令对应的是C语言级别的函数。如果我们产生大量重复的opcode（纯PHP写的代码和函数），对应的则是Zend多次逐条执行这些C代码。而JIT所做的则是更进一步，将大量重复执行的字节码在运行的时候编译为机器码，达到提高执行效率的目的。通常，触发JIT的条件是代码或者函数被多次重复调用。

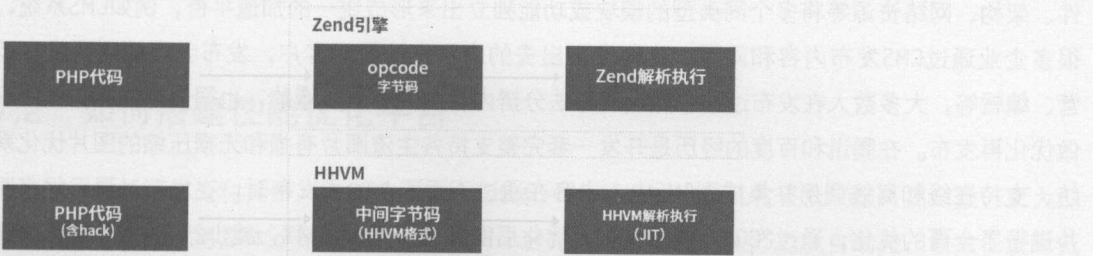


图6-56 HHVM提升PHP性能示意图

第7章 性能优化平台搭建实践

7.1 为什么要搭建优化平台

与搭建自有监测平台意义一样，自有加速平台意义非凡，可以保障用户访问到的内容和应用经过自动优化后都是最佳状态、最快响应，这里的优化平台主要指企业应用架构中的同类型部分抽象出来，形成高性能、高可用的类SaaS平台，例如静态平台、动态平台、数据库平台等。以静态平台为例，大家都知道网页70%的被浏览器加载的内容都是静态的图片、JS、CSS等，能对这些静态内容进行自动内容和网络优化将最大化网页的速度，而且静态平台可以覆盖全公司所有产品，无论哪种产品类型都有静态内容，接入静态平台后速度可以提升100%~200%，部分旧业务接入甚至达到300%以上。

速度与成本优化双重收益

优化平台最重要的追求是快速度、高性能，这一点永远放在第一位的，通过更前沿的硬件、软件、架构、网络资源等将多个同类型的模块或功能独立出来形成统一的加速平台，例如CMS系统，很多企业通过CMS发布内容和网页，这些发布出去的内容直接面对客户，发布者有前端研发、运营、编辑等，大多数人在发布过程中是没有办法分辨内容的大小、快慢的，也很少为体验去做专门做优化再发布。在腾讯和百度的经历是开发一套完整支持各主流图片有损和无损压缩的图片优化系统，支持在线和离线调用并集成到CMS中，内容在通过不同职业的人发布时，在发布过程已经将图片进行了全面的优化，通过2000万张图优化，优化后图片大小减少25.4%，减少25%的图片带宽，速度与成本双重收益。例如动态PHP平台，将分散的新老产品的PHP集中在高性能PHP环境托管运维，

可以下架大量的老服务器及腾挪出对应的机架，也会达到速度与成本双重收益。

平台效应与生产力兼备

往往因历史原因，企业发展过程肯定会留下了稳定型产品、发展中产品，以及开发过程中的新产品，这些不周时期的产品都需要静态、动态、逻辑、存储、计算等公共资源，将这些需求剥离出来形成共同的高性能平台，规模化后就是内部的云平台，而这些平台再经过演进和深度自动化后就是现在大家所熟悉的公有云，例如腾讯、百度、阿里都是通过先满足内部产品需求，再外延实现对外平台化并不断丰富蜕变成为主流公有云。

快速与高性能平台也伴随平台化而产生，例如CDN平台，无论静态还是动态、全站加速CDN，最直接的价值就是加速，其次附价值才是带宽成本减少收益。而整个公司都使用一个CDN平台或动态PHP或MySQL平台时，平台的价值已经发生了质的变化，特别是集中化管理和托管后产生的平台效应和巨大平台生产力将释放对等规模的资源和人力，在大的互联网企业收益是按数千万来衡量的。

接近用户的平台才可持续

在互联网企业中，无论是开发、测试、运维、系统还是基础架构部都有适合自己做的平台方向，但只有接近用户或成为产品的一部分才能可持续，因为只有创造用户价值才更具价值，而速度平台离用户最近，让用户直接受益。例如各大云平台的GSLB、CDN平台、存储、数据库等平台直接影响每次用户请求。而这些平台相比内部平台而言，可以与产品、用户规模一起壮大，具有持续的生命力。

第三方平台不可控性

第三方在初期始终具有它特殊的价值，但规模化后又不能强依赖，无论从成本、服务、数据安全等各方面考虑，唯有自建才可以持续，而且自建经过需求不断打磨后可以与第三方并肩，甚至是超过第三方厂商，例如腾讯在没有自有CDN时主要使用三家第三方厂商，随着使用CDN的产品线和规模越来越大时，巨大的成本和不可控的服务驱使最终自建，进而演变为腾讯云CDN服务，最终对外提供服务。又例如阿里去IOE也是如此，第三方厂商的成本、灵活性、可控性在企业规模化后都将成为瓶颈而非助力。

7.2 如何搭建性能优化平台

工欲善其事必先利其器，搭建强大的性能优化平台可以让性能优化更低门槛、更高效，更大的意义是建立企业级用户体验保障体系，也唯有平台化才可以最大化生产力及自动化，最大程度降低应用性能问题的发生，始终保障卓越用户体验。而性能优化平台伴随在企业整体平台化进程之中，

需要在企业整体平台化过程中并行考虑进去，将个人的经验分享如下。

按通用应用分类平台化

谈到淘宝，大家应该对淘宝的CDN平台有很深刻的印象，近几年双十一最高流量达到数T，是当下互联网公司最大规模的静态应用平台，分析淘宝网站不难发现，淘宝80%~90%的流量是由静态图片贡献的，这个案例可以关联所有的电商类、社区类、资讯类产品等，通常性能优化平台主要有以下四个方向。

- 1 静态类平台，大图、小图、文本、JS、下载、视频等。
- 2 动态类平台，逻辑、列队、消息、推荐、账号、关系、PHP、Java等。
- 3 数据类平台，计算、存储，数据库、日志等。
- 4 工具类平台，GSLB、CDN、TCP加速、速度准入、速度巡检等。

按终端可以分为：

- 1 网络加速，例如CDN加速，可以分为静态、动态、海外、全站、图片、移动加速，BGP加速，例如小运营商网络加速等。
- 2 Web加速，统一高性能前端架构、前端发布自动优化、图片优化等。
- 3 移动加速，Web app、Native app、轻应用、移动API、移动网络整体加速解决方案。
- 4 应用加速，统一PHP、JAVA、MySQL等高性能应用平台。
- 5 系统加速，新服务器、FLASH闪存、SSD加速，TCP加速等。

将对平台化应用做到极致

淘宝的CDN是一个例子，规模、架构、硬件、内容、速度、成本追求极致，这一个典型的应用平台成功案例，这种应用平台可以做到部门级、公司级平台，甚至做到业界最好的平台。平台不是一蹴而就的，从小到大，由粗放到精细，而且不断吸收历史及新产品中的同类应用，不断壮大。传统运维新上线的业务如同需要准备很多原料，再对原料进行加工，而平台化之后，只需要拿组件进行组装即可，而且这些组件不用自己维护，又能保障最佳性能。

另外从平台考虑，运营资源（各区域+各IDC的服务器、带宽、专线、QOS）与产品线的现有架构和实际增量需求（包括未来架构变化，扩展、优化），都与容量管理、成本管理相关，最终驱动预算和预算模型实现，最终又通过运营成本体现。大公司有多多个事业部多个产品，而每一个产品都需要多种运营资源，都需要容量管理、成本和预算管理，也会存在大量性能瓶颈，让每个产品每个

团队都具有好的规划能力、性能优化能力是不现实的，而将产品中的模块分类抽取合并接入到高性能平台，不但可以保障最好的性能外，还可以统一规划和管理，将运营资源进行有效控制。

第三方可以成为一部分

这里说的第三方是指业界卓越的性能优化服务商，例如CDN厂商，各种云服务商（SaaS、PaaS、IaaS），TCP加速等，现在是一个云化的时代，几乎所有云都提供个性化的API服务，即使不能提供也可以通过定制来实现，只需要将这些功能通过API植入到内部平台即可使用，记录这些API的可用性从而达到稳定。特别是中小企业在快速发展过程中，第三方是一个较好的选择，但要能驾驭和引导他们，当然要在预算和成本可以接受的前提下。

优化平台架构及说明见图7-1和表7-1。

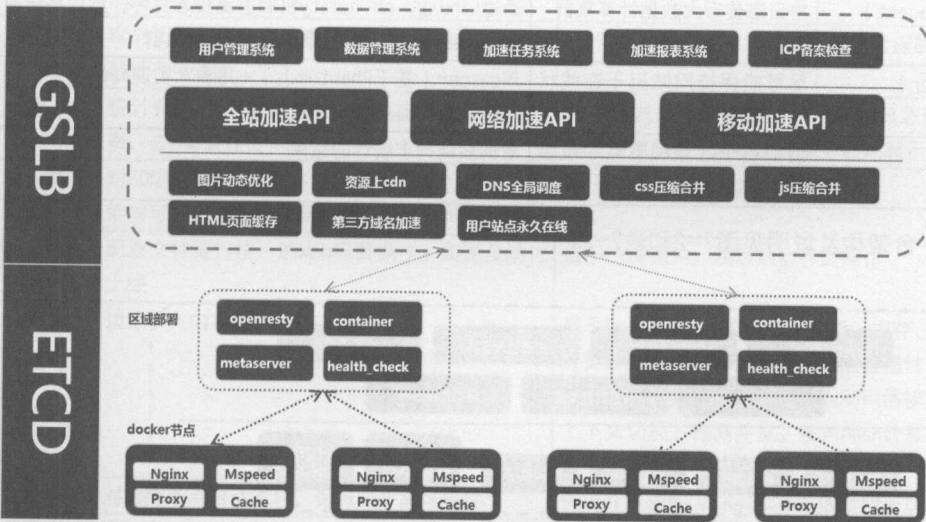


图7-1 优化平台后端架构图

表7-1 优化平台架构模块及介绍

模块	作用	实现原理
回源模块（proxy）	两方面的代理，一个是代理到用户源站，一个是代理到优化后资源的地址	Nginx+PHP，通过修改host和referer，并增加http扩展头，用来控制304返回
优化模块（pagespeed）	对页面进行重构，图片压缩与格式修改，css压缩合并，JS压缩合并，去除页面中多余内容（空行与注释）	Nginx+pagespeed，pagespeed通过异步加载解析页面结构，将页面改写，获取图片资源并优化存入memcached缓存，新版本支持解析第三方域名，并优化改写第三方域名的URL
页面缓存（cached）	对html页面进行缓存	Nginx+proxy_cache，通过正则表达式匹配html文本类资源，将资源存储在本地磁盘缓存

续表

模块	作用	实现原理
任务发布 (etcd)	将增删改查配置的任务发到etcd集群	通过调用etcd集群的API将配置任务发布，etcd会在全网分布式节点中同步配置，保证任务的发布的可用性和时效性
任务发现 (confd)	将etcd任务获取，并同步到节点本地配置文件中，并重启Nginx服务	confd，通过监听etcd中对应节点path下键值的改变，获取对应的配置参数，并通过本地的配置模板将参数写入一个临时文件，对这个临时文件和真正的配置文件做md5比较，如果发生改变则更新配置文件，并重启Nginx服务
动态路由 (openresty)	对某个区域内所有后端节点做负载均衡	Nginx+lua，在redis中维护一份域名与后端节点对应关系的路由表，通过lua加请求随机打到一个后端节点
健康检查 (health_check)	检查后端节点的可用性	Python脚本+crontab，当后端异常时将该节点从路由表中移除
日志上报 (log_agent)	将日志传回大数据计算集群	通过Linux的 on notify机制监测日志文件，然后将新增日志发给kafka，kafka最终传给storm集群
永久在线 (casper)	尽可能保持网站出于在线状态	用casper（基于PhantomJs）+JS脚本实现网站爬虫，将网站常用页面爬到本地做镜像
容量监控	监视各个区域和节点中流量与带宽，并报警	查询后端日志统计的结果

自建CDN平台架构及说明见图7-2和表7-2。

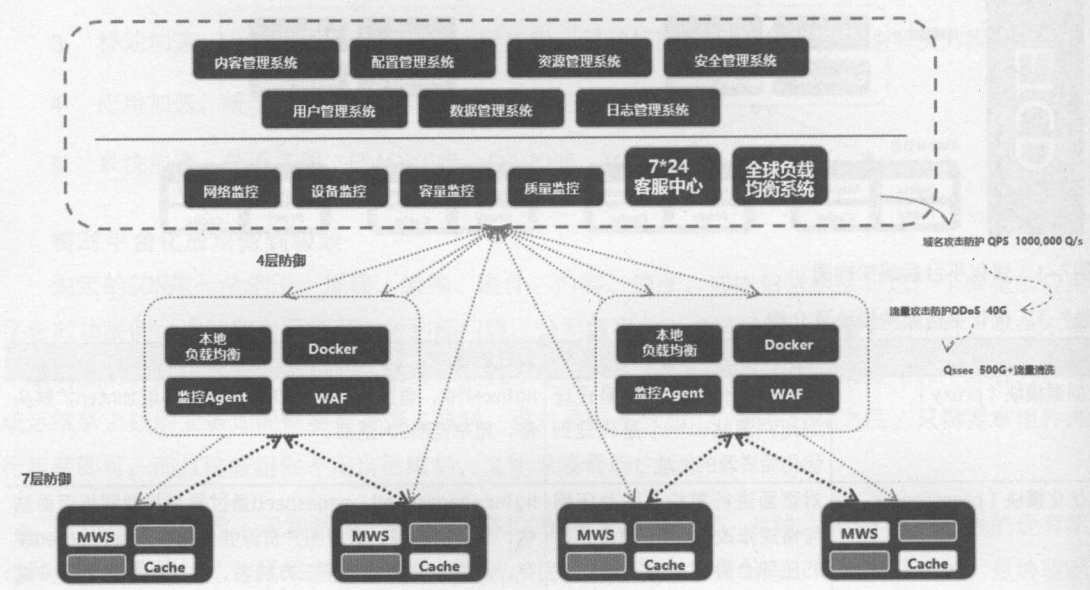


图7-2 自建CDN平台后端架构图

表7-2 自建CDN平台模块及介绍

模块	作用	实现原理
cache	边缘节点内容缓存，就近访问	基于开源软件squid二次开发，使用salt分发配置，实现缓存功能。支持http https协议访问
WAF	Web应用防护系统。预防针对cache服务器发动的攻击。可以有效的防御SQL注入攻击、xss攻击	使用开源软件ModSecurity，针对缓存服务的配置，防御SQL注入攻击、xss攻击
防攻击	CDN平台防攻击系统。保障CDN平台安全、高可用性	基于bind开发的GSLB，实现防御DNS Query Flood攻击的功能。WAF系统防御DDOS攻击、SQL注入攻击、xss攻击
日志	基于原始日志进行流量统计、计费统计、任务状态统计，生成对应报表	自主研发的日志采集器，对日志搜集并上传至存储集群，通过实时计算、离线计算模块进行数据统计
计算	对边缘节点日志进行分析，统计出每个用户不同域名的计费日志和访问日志。对每个域名进行大数据分析，分析出不用运营商、不同地域的用户分布	Kafka集群与Storm集群，zookeeper集群实现实时计算的功能。Hadoop集群实现离线计算的功能
刷新	客户更新和删除源站资源时，将资源从CDN平台的缓存中清除	调用Cache模块的http接口，删除对应的资源
计费	统计带宽费用，给客户结算。包括支持95计费、去3峰值计费	自主研发日志计费模块。对原始日志进行分析提取计算，通过日志采集器上传服务器端展示
设备管理	对CDN平台所有服务器进行统计管理，运维人员能够可视化查看服务器配置、上线时间等信息	基于LNMP架构，通过后台界面操作
配置管理	配置文件进行统一的Web界面操作。实现自动化上线	基于LNMP架构，通过后台界面操作
MWS	边缘节点内容缓存，就近访问	基于Nginx开发，实现缓存功能。内存、SSD、SATA三层存储，根据热度存储。提高磁盘容错性，对只读或者不可读磁盘摘除。多线程事件驱动网络模型，减少线程间上下文切换，提高并发。使用hash计算headerid，实现批量删除缓存功能
负载均衡	对故障节点自动漂移，保障服务高可用性	使用开源软件LVS,Nginx实现负载均衡功能
容量管理	对容量进行评估，保证服务冗余，在服务饱和前提前预警	对硬件资源、带宽、服务并发等瓶颈进行统计，通过算法，估算出服务容量
质量监控	对线上服务质量进行监控	秒级可用性监控，实时发现服务异常。真机监测，监控真实网民访问质量。OS监控，对系统进行体检，包括服务进程、日志、http状态等监控。迅速定位异常，保障服务高可用性

能下降并引起成本上升，再通说就是导致了应用的性能下降。再通说就是应用已经非常复杂，包含多个产品代码或数据流，控制起来花费大量的人力和时间，需要大量的测试和验证才能有效。这种事的控制就是在性能基础上所应的基本的性能规范，需要更多、更好的、更完善的相关的性能规范保障性能的行为规范，通过大家制定的性能规范来保证应用的性能下降并引起成本上升并引起性能下降，从而改善、再通说的性能规范来保证应用的性能下降并引起成本上升并引起性能下降。

再通说就是应用已经非常复杂，包含多个产品代码或数据流，控制起来花费大量的人力和时间，需要大量的测试和验证才能有效。这种事的控制就是在性能基础上所应的基本的性能规范，需要更多、更好的、更完善的相关的性能规范保障性能的行为规范，通过大家制定的性能规范来保证应用的性能下降并引起成本上升并引起性能下降。

第4部分

标准、保持篇

第4部分 应用性能优化标准

第4部分 应用性能优化标准

第4部分 应用性能优化标准

第4部分 应用性能优化标准

第4部分 应用性能优化标准

第4部分 应用性能优化标准

第4部分 应用性能优化标准

第4部分 应用性能优化标准

第4部分 应用性能优化标准

第4部分 应用性能优化标准

第4部分 应用性能优化标准

第4部分 应用性能优化标准

第4部分 应用性能优化标准

第4部分 应用性能优化标准

第4部分 应用性能优化标准

第8章 应用性能优化标准

8.1 防止应用性能退化概述

产品生命周期一直是在变化的，任何互联网企业都是由多个行业细分、多个产品体现商业价值，有新旧、大小之分。互联网产品经过产品策划、设计、研发、运营，然后上线进入市场，它的市场生命周期才算开始。产品、架构、成本都会随时间推移而改变，呈现一个由少到多由简单到复杂的过程，就如同人的生命一样，由诞生、成长到成熟，最终走向衰亡。而不同阶段对性能的要求也是不一样的，产品初期和衰退期对性能要求较低，产品成长期和成熟期对性能要求较高，整体在整个产品生命周期中大部分阶段对性能和用户体验是有要求。

往往产品在迭代过程中性能会发生较大变化，比如新版上线，哪怕是一个网页元素变大都会原来的性能基础上增加加载时间，例如服务器折旧、人员流失也会影响应用性能。特别在之前从事过性能优化的产品，如果在往后的产品迭代中不加以管理，将会使之前优化的劳动成果付之东流，即使及时再优化，也很难保持。所以非常有必要从流程、规范、平台、告警等维度融入到产品生命周期中，对当前应用性能进行保持，防止应用性能退化，从而使优秀的应用性能不断随产品迭代而保持并良性循环。

8.2 通过规范防止性能退化

应用性能是一个动态变化的过程，例如在BAT大的产品一天迭代上百次，而在产品迭代的过程中，经常会出现性能抖动，往往这种抖动由迭代版本的差异导致的。如果在迭代过程产生了应用性

能下降并已经发布出去，再通过性能监测平台追踪到性能下降，再去优化解决问题已经非常滞后了，而且多个产品迭代再反复发现、修复将浪费大量的人力和时间，需要有事前机制来规避是最有效的，这种事前机制就是在性能基准上形成的基本的性能规约，需要研发、测试、运维等所有相关的角色确立的保障性能的行为标准，通过大家制定的性能规范避免让应用性能下降及对速度优化成果进行可持续的保持。因网络层、系统层的随常规版本迭代变化影响较小，所以性能规范的主要范畴是生产环境中的内容、应用、架构变化及对应的研发和发布负责人。

用户感知维度见表8-1。

表8-1 用户感知规范参考表

类别	快	较快	慢	很慢	性能指标
用户敏感	<1s	1-1.5s	1.5-2.5s	>2.5s	首屏时间、白屏时间
用户不敏感	<2s	2-4s	4-8s	>8s	整页时间

最佳：白屏<1s，首屏<1.5s，onload<3s

核心指标维度见表8-2。

表8-2 核心性能指标规范参考表

网络层指标	好	正常	差	备注
可用性	大于99.99%	99%~99.9%	小于99%	反映页面打开成功率，过低影响站点形象
DNS	小于0.18s	0.18~0.3s	大于0.3s	DNS解析时间，用户访问页面的第一步
建立连接时间	小于0.15s	0.15~0.3s	大于0.3s	建立连接时间是指 IE 浏览器和WEB 服务器建立 TCP/IP 连接的消耗时间
重定向时间	无	小于0.1s	大于0.1s	重定向时间是从收到 WEB 服务器重定向指令到请求 WEB 服务器的第一个元素之前的消耗时间
首包时间	小于0.2s	0.2~0.4s	大于0.4s	第一个包时间是指浏览器发送 HTTP 请求结束开始，到收到WEB 服务器返回的第一个数据包的消耗时间
总下载时间	小于10s	10~20s	大于20s	总下载时间是指页面所有内容下载时间，总下载时间主要表示的是页面的总体耗时，不同类型站点标准不同。现有标准仅仅作为默认参考
SSL时间	小于0.3s	0.3~0.6s	大于0.6s	SSL安全认证时间，反映认证应用性能
客户端时间	小于0.3s	0.3~0.5s	大于0.5s	浏览过程中去除网络层的时间后IE本地渲染的时间

前端开发维度

前端决定了浏览器加载的内容和结构，也是性能问题的主要生产者，如果前端团队在前端性能规范的框架下，写高性能代码，有好的编程习惯，将有益于用户体验。然而，对于构建大型Web应用的团队来说，要坚持贯彻这些规划并不是一件十分容易的事。因为规范中很多要求与开发效率、习惯相违背，特别在初期会严重影响团队成员间并行开发的效率，尤其是在团队有版本管理的情况

下，每天要花大量的时间进行代码修改合并，这项成本是难以接受的。因此在前端工程世界，总会看到周期性的性能优化工作，但只要团队形成和遵守优秀的开发习惯和规范后，开发质量、效率和用户体验都会得到平衡，这是最理想的状态。前端规范可以参考表8-3。

表8-3 前端规范参考表

规范	规范说明	备注
图片优化规范	<div>1. 页面图片必须经过质量压缩才能使用（推荐质量压缩比：60%~70%），不要在页面中直接写入图片。</div> <div>2. 重点页面（首页、频道首页、底层页）要求div+CSS布局，同时做素材合并。</div> <div>3. 基础项目产品的页面要求要求div+CSS布局，同时做素材合并</div>	图片占网页大小的60%~80%，如果不注意，一张图片可以将速度拖慢50%
CSS优化规范	<div>1. 一个页面原则上只允许有一个CSS文件，且CSS文件放在页面<head>内。</div> <div>2. 页面内联的css代码，原则上放置在页面<head>内。</div> <div>3. 页面布局不能使用超过900px高的table。</div> <div>4. 素材合并：对于超过5屏的页面，素材应该合并成多个文件，例如：前3屏的素材合并成一个文件，中间3屏的素材合并成一个文件，最后3屏的素材合并成一个文件</div>	CSS是网页的灵魂，永远要确保你的css文件是干净、简洁的代码
JS优化规范	<div>1. 页面首屏尽量不要出现JS文件请求。</div> <div>2. 页面通用功能（例如：焦点图、flash加载、图片滚动、文字滚动、投票调查、订阅、页卡切换、异步加载等）要求使用静态域名下全站统一的JS文件。</div> <div>3. 任何JS功能要求可以做到延迟加载，避免阻塞页面。</div> <div>4. 超过10K的JS代码不允许直接写入页面内，应该使用外联JS文件。</div> <div>5. 不允许在页面中使用document.write直接输出页面代码。</div> <div>6. 禁止在首页、频道首页、底层页引用体积庞大的JS公共库（例如：prototype、jquery、trimpath等），应该只使用精简过的JS库代码。</div> <div>7. 公共产品提供的接口JS代码注意封装好自己的命名空间，避免代码命名冲突，且严格控制JS文件大小和个数，通用要求：公共产品只能引用1个不超过20KB的JS文件</div>	JS是公认的网页性能杀手，JS数量和大小、逻辑、位置都可以影响网页首屏时间、总下载时间，任何产品形态，都要对JS特别关照，精益求精的进行优化
广告代码优化规范	<div>1. 普通广告加载方法：首屏之后加载首屏内的广告代码及素材，第二屏之后加载第二屏内的广告代码及素材，依此类推。</div> <div>2. 富媒体广告加载方法：富媒体广告代码必须在页尾加载，广告出现必须在页面全部加载完成之后。</div> <div>3. 第三方广告代码加载与上述两种方式相同，必须要求第三方代码支持这两种加载方法</div>	互联网离不开广告，部门内、跨部门及第三方广告都可以轻而易举举让之前的优化白费
Iframe优化规范	<div>1. 尽量少使用Iframe页面，原则上：内部内容不能使用Iframe页面，外部内容尽量利用抓取系统内容抓取到服务器上。</div> <div>2. 复杂功能的第三方Iframe页面必须经过优化组审核并优化通过之后才能上线</div>	最好不要用Iframe
统计代码使用规范	<div>1. 一个页面只允许包含一次统计代码，Iframe页面内不允许包含统计代码。</div> <div>2. 统计代码只允许放到页面首屏之后，不允许放到首屏内，首屏高度定义为（900像素左右）</div>	统计代码也是主要的性能杀手，特别是国外第三方的
第三方代码优化规范	<div>1. 尽量少直接使用第三方页面或代码，且必须遵守公司制定的第三方页面代码使用安全规定。</div> <div>2. 简单页面可以直接通过发布系统抓取功能实现。</div> <div>3. 第三方页面必须遵守上述“图片、CSS文件、JS文件优化”对页面进行优化</div>	最好不要嵌入第三方应用

续表

规范	规范说明	备注
公共平台使用规范	1. 页面服务器集群上只能存放文本文件（即页面、css文件、JS文件），不能存放图片媒体文件（即图片、flash、mp3等）。 2. 素材服务器集群上主要存放素材文件，包括素材图片、css文件、JS文件、mp3、flash等。 3. 图片服务器集群上主要存放内容图片，不允许存放其他内容	公共平台是按文件类型设计最优架构和网络环境，如果交叉合作会让性能下降，甚至是不可用

8.3 通过流程防止性能退化

互联网企业产品设计、研发、测试、发布、运维过程中，也伴随性能问题的发生，需要建立一种机制来保障用户体验，而这种机制需要依附在产品的生产过程之中，需要所有环节参与的角色达成规约，通过共同建立科学合理的工具、基准并执行达到保障和改善用户体验的目的，通过腾讯、百度的大规模性能优化经验，主要有三种，如表8-4所示。

表8-4 防止性能退化流程及对比

流程	作用	优势	劣势
性能准入	在发布之前过滤掉存在的性能问题，保障发布质量	事前发现，主动解决问题	依赖人为参与，受益于准入机制建立之后，覆盖不了之前发布的存量应用
性能认证	针对生产环境核心产品同终端和类型的应用进行性能认证排名，长期跟踪应用性能，防止性能退化	长期跟踪，设立基准并增强整体质量和团队能力	需要同类，同属性的前提条件，例如页面、Web App
性能巡检	针对发布之后，存在于生产环境的所有应用进行大规模性能检查，将通用、潜在的性能暴露出来，优先修复长尾性能	大批量，覆盖范畴广，甚至能过挖掘社区用户抱怨发现性能问题	因规模大，主要偏可用性监测，发现大颗粒度的性能问题，辅助真实用户性能数据

8.3.1 应用性能准入

无论是主动监测还是被动监测，监测任务都是需要指定并且是能被用户访问到，以真实用户的场景反向分析应用性能存在的问题，这种方式以真实用户体验为基础去做优化也是我们做应用性能优化最大的出发点，因为用户访问的全过程及各环节存在的问题都可以分析出来，但这种方式最大的缺点是滞后，生产环境存在的性能问题已经真实对用户造成了影响，有的甚至从上线以来就一直存在。从腾讯、百度做性能优化的经验看，生产环境存在的性能问题永远比我们关注的核心产品重要模块的性能问题要多很多，只是核心产品优先级更高，即使经过较长一段时间做完性能优化并达到预期，要想达到保持也是非常困难的。

所以需要一种防御为主的准入机制，在应用发布之前，对存在的应用性能问题进行拦截，保障

发布出去的应用是合格的，能给用户带来好的用户体验，这项工作的意义是巨大的，需要研发、测试、运维等多团队配合。在上线前对应用性能进行检查预判，对于存在性能问题的发布在上线前就暴露出来，并进行优化修复，保障发布出去的应用是无损用户体验的。

性能准入基准

性能准入建立在产品发布流程中，需要研发、测试、运维遵守共同建立在应用性能基准之上的规约，简单理解超过基准线不能发布。企业、部门或产品不同，制定的基准是各不相同的，需要具体对待。例如网页类产品首屏时间不超过1s、网页元素不超过30个、页面大小不超过500KB等等。看上去是简单的基准，但要是放在CMS内容发布系统意义完全不同，例如门户网站，几百位编辑发布的内容是完全不可控的，通过设定准入基准，可以轻松将超标的内容挡在发布之前，加上内容的自动优化可以将数百名对应用性能没有任何感知的编辑发布的内容变为可控。

性能准入环境

准入的基准是建立在准入环境之上，用什么数据源做为基准判断是根本，网页相对移动应用稍简单，性能准入环境及实现原理汇总如表8-5所示。

表8-5 准入类型及分析表

终端\实现方式	真机	模拟	备注
PC	基于全国多浏览器、多运营商真实招募用户监测。招募用户有偿安装监测客户端，服务端可以向监测客户端下发监测任务，监测任务发生在真实用户PC及浏览器上，覆盖主要省份，同时在线监测点2000左右	在测试环境部署多浏览器，多监测节点，同时在线监测点要求固定，建议100个左右	PC真机相对模拟监测来说，是完全真实的用户和体验数据，全流程性能数据都可以得到；模拟监测相对单纯，非真实用户体验数据，有衡量作用
移动Web App	不同制式、不同厂商的真实手机。真实网络环境的波动性非常大，测试结果受网络波动影响，而这种波动会掩盖真实的速度变化，而且在这种波动的影响下，难以制定统一的准入标准	Linux服务器上使用phantomJS无显示浏览器，可以实现对页面的自动化加载，并记录下页面的加载过程，在关键时间点进行相关页面截图等动作	服务器上模拟移动网络环境，利用了Linux下流量控制器TC实现网络关键因素的限制，包括延迟，带宽以及丢包率，得到的数据相对稳定，测试移动应用在慢网络下的表现较好
移动Native App	不同制式、不同厂商的真实手机，APP通过服务端下发到手机，并记录下手机端所有数据，例如启动时间，资源占用，耗电量，与后端交互时间等	通过手机模拟器，目前支持主流手机操作系统及主流版本，限于模拟器兼容性，只能做简单通用测试	Native App因期独立和特殊性，目前还没有好的方法实现性能准入，Hybrid App相对容易不少

8.3.2 应用性能认证

与其说应用性能准入是一张质量滤网，把应用性能问题规避在发布之前，那速度认证就是一把

“尺子”，衡量发布之后生产环境中各应用的性能优劣。通过沉淀、固化应用性能监测、优化成果，输出应用性能基准，通过对生产环境中的应用性能持续监测及数据建立性能排名和认证，通过不断跟踪生产环境中的应用性能，从而有力推动产品应用性能健康有序发展，性能认证是建立在持续监测及数据基础之上建立的性能基准，根据企业、产品、部门、团队关注不同，建立的基准各不相同，例如前端团队更看重网页渲染指标，运维、测试团队关心系统及应用指标，网络团队更关注网络指标等。通过建立性能认证，将性能监测、优化升华到新的高度，主要意义如下：

- 1 性能认证排名通过数据驱动可以快速看出多产品各维度性能差距，比一比就知道，无论看综合评分，还是前端加载时间、后端响应时间、网络延时等，可以从各维度看出差距，快速找出优先优化点进行改进。
- 2 提升各产品整体性能水平，促进各部门、产品线交流，快的产品可以不断追求极致，慢的产品可以通过交流和追赶不断向优秀看齐。
- 3 通过性能认证可以驱动产品线对应用性能的投入，特别对于排名靠后的产品线，都是公司产品，都是相同的硬件、网络资源，都是类似团队，是没有理由逃避性能差的。
- 4 性能认证推动性能监测、分析、优化可持续发展，性能认证这把“尺子”可以持续丈量公司各产品应用性能，各部门及各产品可以通过性能认证为自己产品设立基准，通过保持和超越基准打造卓越用户体验。

性能认证还有一些更高级的存在方式，例如：

- 1 性能大赛，通过每年或每季度组织性能优化比赛，可以将公司优秀的性能优化实践和人才聚集在一起，互相融合并取长补短，提升企业和部门用户体验文化和技术氛围。
- 2 周报&预警，应用性能排行榜可以通过邮件的方式，每周甚至是每天发送给相关的负责人及领导，随时关注性能的变化。

8.3.3 应用性能巡检

性能巡检与性能认证类似，前提条件都是发生在发布之后的生产环境，不同之处在于，性能认证受限于排名对比，需要建立在共性的基本之上，适合同类终端产品及系统、网络等共同类资源性能，例如网页性能、移动Web app性能、网络性能等。而性能巡检没有限制，范围更广，例如同时监测数千个核心网页的性能并跟踪网页的可用性、是否存在优化空间等，同时还有以下作用：

- 1 性能巡检在生产环境大批量抽样检查，不是以某一个页面或模块性能指标来衡量，而更多的体现整体性能状况，突出最慢长尾，从而更聚焦这部分慢的应用进行优化，从而提升整

体用户体验。

- 2 性能巡检受益于周期性检测量大及检查的信息丰富，可以从可用性、可优化项、报错、兼容性等多维度进行检查，特别针对生产环境存量的老业务有较大的用户体验提升作用，大量减少人工参与。
- 3 性能巡检弥补性能准入需要人为参与及性能认证需要同类属性制约，可以更粗放的针对大批量的页面、应用API、Web App、服务器、网络等进行定制化的性能检查，将性能监测覆盖范围最大化。

8.4 业界优秀企业的经验

国内外优秀互联网企业在产品性能优化领域做了长期且大量的实践，并将实践成果不断提炼和分享，同时还开发了大量优秀的性能优化工具，其中Yahoo!、Google最为体系并产生了巨大影响，不仅为互联网行业产品迭代提供了性能优化依据和规范，而且推动了性能优化技术、工具的演进。

8.4.1 雅虎Web优化最佳实践

雅虎的Exceptional Performance团队为改善Web性能带来最佳实践。他们为此进行了一系列的实验、开发了各种工具、写了大量的文章和博客并在各种会议上参与探讨。最佳实践的核心就是旨在提高网站性能。Exceptional Performance 团队总结出了一系列可以提高网站速度的方法。可以分为7大类34条，包括内容、服务器、Cookie、CSS、JavaScript、图片、移动应用等7部分。因雅虎性能优化的34条黄金守则提出较早，也广为被熟知，就不在此详细赘述。

8.4.2 谷歌Web优化最佳实践

谷歌Web优化最佳实践与雅虎的类似，也集成到了一个优化工具，名为Page Speed。Page Speed在运行时分析服务器配置和服务器上下载下来的Web代码，还会输出一个分析报告，其中包括如何改进网页的建议，详细内容如表8-6所示。

表8-6 Google Web优化最佳实践表

规则名称	含义
避免出现错误的请求	去掉“坏链”，避免请求资源返回404/410错误，减少不必要的请求
请勿在元标记中指定字符集	在meta标签中设置字符集，将使IE8中的预先下载器（lookahead downloader）功能失效，延长页面展现时间

续表

规则名称	含义
避免在 CSS 中使用 @import	在一个外部样式表中使用CSS @import, 将导致页面加载的额外延时。使用了CSS @import, 将导致浏览器无法并行的下载CSS样式表, 增加了整体页面加载时间
避免使用目标网页重定向	当重定向发生时, 增加页面加载的延时, 并且无法向用户展现页面。在大部分情况, 都需要在不改页面功能的基础上避免重定向的发生
避免使用运行时间较长的脚本	避免需要运行时间较长的脚本用来绘制页面, 这样会降低浏览器的响应
暂缓 JavaScript 解析	为了加载页面, 浏览器必须解析所有<script>标签, 这样会导致额外的页面加载时间。尽量介绍页面渲染需要用到JS文件大小, 当某些JS需要被执行的时候再去解析, 从而减少首次页面加载的时间
启用压缩	许多网络服务器可以通过调用第三方模块或使用内置程序将文件压缩为Gzip格式, 然后再发送该压缩文件以供下载。这样可以在下载呈现网站所需的资源时, 为您节省一些时间
启用 Keep-Alive	开启HTTP Keep-Alive功能, 将允许浏览器使用同一个TCP连接来接收/发送多个HTTP请求, 可以降低后续请求的延迟
内嵌小型 CSS	当外部CSS文件较小时, 可以内嵌这些CSS文件, 节省了浏览器去下载这些文件的开销
内嵌小型 JavaScript	当外部JavaScript文件较小时, 可以内嵌这些JavaScript文件, 节省了浏览器去下载这些文件的开销
使用浏览器缓存	如果用户会多次访问你的网站, 那么静态资源的浏览器缓存可以节省用户的时间。缓存标头应当应用到所有可缓存的静态资源中, 而不仅仅是应用到一小部分静态资源(例如, 图片)中。可缓存的资源包括JS和CSS文件、图像文件及其他二进制对象文件(媒体文件和PDF文件等)
压缩 CSS	压缩CSS文件大小, 有助于加快下载、解析以及执行的时间
压缩 HTML	压缩HTML文件大小, 有助于加快下载、解析以及执行的时间
压缩 JavaScript	压缩JavaScript文件大小, 有助于加快下载、解析以及执行的时间
尽量减少重定向	尽量减少HTTP重定向, 这样可以节省不必要的请求往返时间以及用户的等待时间
尽量减少请求的数据量	保证请求的Header和Cookie足够小, 确保一个HTTP请求可以装在一个TCP包内。目前使用最广泛的包大小限制为1500B, 所以需要尽量使一次HTTP请求大小小于1500B
优化图片	通过加载合适大小的图片, 降低页面的加载时间
优化样式表和脚本的排列顺序	正确的外部样式表以及外部/内嵌脚本可以增加下载的并发度, 加速浏览器渲染时间
首选异步资源	异步加载那些会阻塞页面加载的资源
将 CSS 放在文档标头处	将样式块代码以及<link>元素从body移至头部, 加速页面渲染性能
将查询字符串从静态资源中删除	大多数代理缓存系统, 例如Squid 3.0, 将不会为包含"?"的URL创建缓存, 即使返回header中包含Cache-control: public。为了让代理可以缓存这些资源, 需要将URL中的query string去掉
由同一网址提供资源	只从一个唯一的URL地址获取一个资源, 避免重复的下载以及多余的网络往返时间

续表

规则名称	含义
提供压缩后的图片	适当的压缩图片可以将资源大小降至最低
改善服务器响应时间	长时间的服务器响应时间会拖慢页面加载时间。PageSpeed希望服务端可以快速返回请求，使得页面加载足够快。对于每一个请求的资源，服务器需要从发送首字节开始在200ms以内返回数据
请指定缓存验证工具	所有静态资源都应该包含 Last-Modified 或者 Etag 头部信息，这样可以最大化利用浏览器的缓存机制
请指定一个“Vary: Accept-Encoding” 标头	一些代理缓存系统中的bug会导致向不支持压缩的浏览器返回压缩版本的资源。指定了“Vary: Accept-Encoding” 头可以显示让代理缓存系统保留压缩和未压缩两个版本的资源
指定字符集	在服务端设置字符集，降低浏览器的处理时间
指定图片大小	为所有图片指定宽度和高度，这样可以消除不必要的图片重绘，进而加速页面渲染
将图片组合为 CSS 贴图定位	利用CSS贴图定位 (CSS sprites) 尽可能降低图片文件数量，这样可以减少往返数据的次数、下载其他资源的延时，降低资源请求负载，并且可以减少整体网页的数据量

第9章 应用性能优化保持

9.1 性能优化保持概述

性能优化保持是性能优化中、后期的重要工作，性能优化一定意义上是周期性工作，优化到最佳状态后，如果不进行防范，性能随日常迭代会出现退化，就如很多人减肥一样，如果减肥成功后，运动和饮食一放松，体重会快速反弹，性能优化保持也类似，所以性能优化之后需要通过一系列的巩固措施让之前性能优化收益持续下去。

9.2 通过平台防止性能退化

百度、腾讯、淘宝等各大公司在多年性能优化最佳实践的基础上做了一系列提升产品性能与开发效率的工程化方案，将这些方案集成到各产品平台，让多个产品团队的性能优化通过这些平台自动化完成和规避。这些平台主要包括自动优化开发框架与平台两部分，可以大量减少人工参与性能优化的成本和风险，提升了性能优化各团队协作效率，进而来达到更快、更可靠、低成本的自动优化目标。

9.2.1 自动优化开发框架

这里所说的开发框架更侧重前端，因为前端决定了用户终端加载的绝大部分内容及加载逻辑。Facebook、Twitter及国内BAT等优秀企业都通过开发框架自动优化性来保持和提高每一次新发布的

性能，并且能够给出精确的性能评估，避免产品迭代过程中出现性能退化。开发框架能自动应用Web性能的最佳实践，通过自动分析和修复相应的性能问题来帮助提高网站性能，可以无缝接入持续集成流程中，在产品研发过程中保障性能不出现退化，避免体验问题的产生。自动优化开发框架示意图如图9-1所示。



图9-1 自动化开发框架示意图

9.2.2 自动优化基础平台

静态资源包括JavaScript、CSS和图片、视频等，它们是现代网站不可缺少的部分，而且这些静态资源占网页大小的比例达到70%~80%，有的甚至更高，显而易见这些资源及对应的网络是造成产品体验问题最直接的原因。无论是Web性能优化最佳实践，还是IDC、CDN网络加速，最终都是通过改变网页内容及结构并加快网络传输效率达到提升用户体验的目的。而这两种大的优化方向都需要多个专业的人或团队花时间和精力来完成，而且较大概率会出现性能退化。所以需要自动化的基础平台来完成这两部分工作，不但可以减少人力，还可以准确、高效、可持续的自动化完成优化，目前主要有以下三种基础平台。

- 1 图片优化，根据HTTP Archive的统计数据，图片平均请求占静态资源请求的66.2%，平均大小占静态资源的70.3%。通俗的理解就是用户访问网页时，看到的绝大部分内容是图片，通过封装各类图片优化API，各产品线可以自由选择在线、离线图片优化，优化后的图片直接推送网络加速。
- 2 网络优化，网络两端，一端是加速的内容（所有种类的静态资源，部分动态应用），一端是网络资源（国内、外所有种类的网络资源），两端通过域名“连接”，各产品线可以DIY自助选择加速内容。
- 3 全站优化，整站加速，全站托管，通过建立最优内容优化并结合最优硬件、网络、系统于一体的自动化全站加速，只需要域名解析授权，整站内容自动进行优化，并推送到最优网络、系统环境，并实现永久在线，自动容灾。

9.3 通过告警防止性能退化

在进行性能优化的过程中，会逐步构建起了应用性能监控平台，来收集真实用户的性能数据、

挖掘性能瓶颈、评估优化效果，该平台监测数据也会越来越多，也会被越来越多的产品线使用。而应用性能是一个持续性的工作，即使是监控最完备的应用也会出问题，因为对于每一个发布周期可能带入的性能退化。所以完善的告警机制是非常有必要的，包括个性化的告警策略、多样化的告警渠道以及智能的告警过滤、告警合并与关联功能。

告警策略

提供个性化告警策略，包括默认告警策略、自定义告警策略等。

- 1 默认告警策略包括首屏时间、响应时间、网络情况、错误、内存、CPU使用率、错误率、崩溃率等基本性能指标。
- 2 自定义告警策略则可设置更多个性化的性能指标告警策略，如设置某一类型监测的某个交互响应时间阈值等。

告警渠道

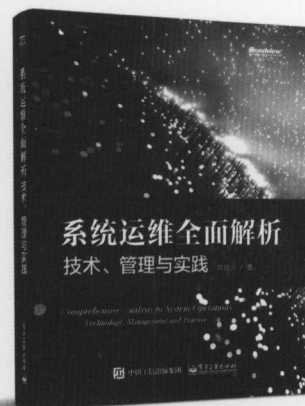
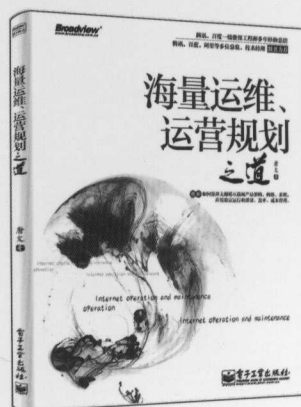
提供多样化告警渠道，可以根据需求，选择不同的告警渠道。

- 1 邮件：通过邮件的形式发送告警。
- 2 短信：通过短信的形式发送告警。
- 3 推送：结合手机特性，降低运营成本，通过移动App推送告警。系统发出告警后，通过点击“告警内容”，可直接跳转至该条告警的详细信息。
- 4 微信告警：通过微信发送告警，帮助随时“掌握”应用性能。

告警过滤、告警合并与关联

- 1 告警过滤：对于告警信息，根据用户设定的过滤条件，屏蔽部分告警（如根据用户设置，忽略某个特定URL请求的网络告警）。
- 2 告警合并与关联：通过智能事件-事件关联分析，避免重复发送告警。

好书力荐



大型网站性能 监测、分析与优化



唐文老师在大型网站运维和性能优化领域有着非常丰富的经验，先后在百度、腾讯参与运维及性能优化基础设施建设。与他共事时，我被其专业素养、产品意识、分享精神所折服，在交流合作的过程中从他身上学到了很多性能和运维方面的知识。在百度工作时，他就有将多年的性能优化经验汇总成书的想法，经过几年的努力，idea 终于兑现了，非常难得。性能是大型网站的命脉，与用户体验、服务稳定性息息相关，也能为企业节省运作成本。这本书涵盖了性能优化的方方面面，包括为什么要做性能优化；如何评估与监测应用及服务的性能；常见的性能优化手段；如何防止性能退化等。书籍理论与实践并重，而且有案例分析，是一本难得的性能优化指导书籍。

百度前端性能优化负责人 牛尧

应用性能管理当前已成为互联网技术圈非常火爆的话题，其内容涵盖了与用户体验息息相关的系统监测、网络服务、应用代码、性能优化等环节。唐文先生在互联网公司的长期性能管理从业经验为本书注入了科学化、系统化、专业化的应用性能管理实践视角。随着移动互联网的不断发展壮大，国内的应用研发体系也在不断健全并趋于成熟化，应用性能管理将逐渐成为应用研发体系中不可或缺的一环，并为终极的用户体验保驾护航。衷心希望本书能够引导并帮助国内的开发者养成更系统化地应用、网站研发意识与习惯。

阿里巴巴集团高级技术专家 杨镔，花名冷茗



博文视点Broadview



新浪微博
weibo.com

@博文视点Broadview

上架建议：计算机\大型网站

ISBN 978-7-121-29142-5



9 787121 291425 >

定价：88.00元



策划编辑：张月萍
责任编辑：徐津平
封面设计：吴海燕